EP34188 (5) a)

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/44 | A2

(11) International Publication Number: **WO 99/63431**

(43) International Publication Date: 9 December 1999 (09.12.99)

(21) International Application Number: PCT/US99/12075

(22) International Filing Date: 28 May 1999 (28.05.99)

(30) Priority Data:
60/087,440 1 June 1998 (01.06.98) US

(71) Applicant *(for all designated States except US)*: QUALITY CONSULTANTS, INC. [US/US]; Suite 215, 1775 The Exchange, Atlanta, GA 30339 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: BROCK, Kevin, R. [US/US]; 1277 Peninsula Trail, Lawrenceville, GA 30044 (US).

(74) Agent: BERNSTEIN, Jason, A.; Bernstein & Associates, P.C., Suite 121, 30 Perimeter Center East, Atlanta, GA 30346-1902 (US).

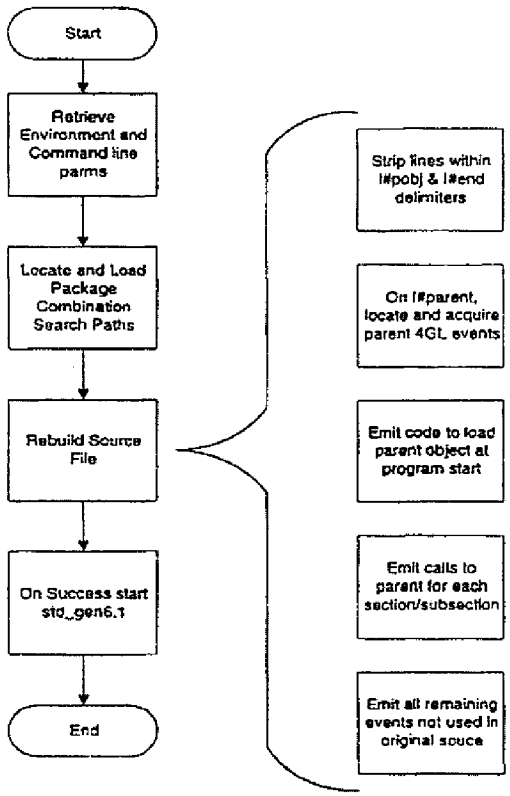
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: PREPROCESSOR FOR ENCAPSULATING SOFTWARE SOURCE SCRIPTS

(57) **Abstract**

In BAAN software, a user defined script is created by copying a standard script to a custom directory and making the required changes to the standard script so as to obtain a custom script. First, the user creates a custom script containing only the user's add-ons and modifications to the standard script, whereby preprocessor directivess are used to call or override code contained in the standard script. Second, the QKEY preprocessor uses the directives to automatically add to the custom script the BAAN software instructions necessary to make the appropriate calls and to dynamically load the standard script. Finally, the user script is compiled, using only the standard script's object code (not the source code) to resolve the calls. The preprocessing is performed without having the standard script's source code available. When the standard source is patched, the custom script will just need to be compiled. Very similar to the concept of inheritance in OO programming, whereby a child class (custom script), derived from a parent class (standard script), overrides or inherits the functionality of the parent.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

PREPROCESSOR FOR ENCAPSULATING SOFTWARE SOURCE SCRIPTS

FIELD OF THE INVENTION

The present invention relates to a software program for preprocessing source scripts of existing software.

BACKGROUND OF THE INVENTION

Software has been developed by a company called Baan. One application of this software is in enterprise resource planning ("ERP"). Baan® software contains source scripts. the source scripts written by the en user will be derived from a Baan® standard object. This derivation is dynamic so, if the Baan® standard object changes (i.e., through a patch), the current user object will automatically receive these changes. This is known as "encapsulating" the standard object so changes to it do not necessarily require changes to the additional code added by the end user.

SUMMARY OF THE INVENTION

preprocessing existing source scripts, comprising: (a) a computer having a memory storage device and a microprocessor; (b) an operating system stored in said memory storage device; (c) means for viewing an object; and, (d) means for permitting the overriding of the associated events of said object.

Accordingly, it is an object of the present invention to provide a preprocessor for source scripts without having the source code available.

Other objects, features, and advantages of the present invention will become apparent upon reading the following detailed description of embodiments of the invention, when taken in conjunction with the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The various features and advantages of the invention will be apparent from the attached drawings, in which like reference characters designate the same or similar parts throughout the figures, and in which:

Fig. 1 is a flow diagram entitled ENGEN Compilation Time Enhanced Process Flow Comparison (Standard Baan vs. ENGEN Enhanced),

Fig. 2 is a flow diagram entitled ENGEN Run Time Comparison (Standard Baan vs. ENGEN Enhanced), and

Fig. 3 is a flow diagram entitled ENGEN Logic Overview.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In general, the present invention provides software for preprocessing source scripts. Details of the invention are set forth in the following documents attached hereto and incorporated herein: (1) User and Technical Guide, and (2) the code in the attached QSET™ for Baan Tools 6.1 and later and (3) a brochure entitled "Unlock the full potential of your Baan ERP system with

The user of the present invention can preprocess source scripts without having the source code available. This is possible by allowing the user to view each Baan standard session as an object, and by permitting the overriding or extension of the object's associated events (or "public events" in object oriented programming terminology). The result is that a user can create customized 4GL objects that contain only the user's new code so that when a Baan patch or version release control ("VRC") is installed the changes can be reused automatically. In most cases, there is no need to recompile customizations. Even in installations that have already purchased Baan's source code, these features can save considerable time and money.

Among the advantages of the present invention are that there is significantly lowered programming costs, ease of implementation and reduced development time.

Although only a few exemplary embodiments of this invention have been described in detail above, those skilled in the art will readily appreciate that many modifications are possible in the exemplary embodiments without materially departing from the novel teachings and advantages of this invention. Accordingly, all such modifications are intended to be included within the scope of this invention as defined in the following claims.

It should further be noted that any patents, applications or publications referred to herein are incorporated by reference in their entirety.

QKEY™
Encapsulated Generator for Baan®

Version 2.10

User and Technical Guide

April 12, 1998

Table of Contents

Legal Notice 1
Introduction 1
Requirements 1
Supported Environments 1
Release History 2
Version 2.10 2
Version 2.00 2
Version 1.22 2
Version 1.21 2
Version 1.20 2
Version 1.10 3
Upgrading from v 1.x to v 2.x 4
Upgrading from v 1.10 to v 1.20 4
Installation in Unix 5
User Configuration in Unix 7
Concepts 9
Objects as DLLs 9
The Child Object 9
The Derivation Process 10
Using QKEY 11
Debugging 13
Reference 14
|#parent [object] 14
|#inhibit 14
|#call 15
|#pobj 15
|#end 15
Benefits 16
Limitations 16
Examples 17

Legal Notice

Introduction

The QCI Key Developer (QKEY) for Baan is designed to help the end user develop add-on products and enhancements for the Baan system. This tool works as a preprocessor for Baan source scripts. The source scripts written by the end user will be derived from a Baan standard object. This derivation is dynamic so, if the Baan standard object changes (i.e. through a patch), the current user object will automatic receive these changes. We call this "encapsulating" the Baan standard object so changes to it do not necessarily require changes to the additional code added by the end user.

QKEY is very easy to install and use. There are some limitations (see Limitations section), however, it does not limit any current development capability within Baan. QKEY may be used with or without standard source code.

Requirements

QKEY 2.0 and later, is now written in C so will be more generally available. However, because of the nature of compiled languages, it may not be available on all platforms.

To be able to use QKEY you must be using Baan Tools version 6.1 or later. Earlier versions did not use dynamic link libraries for 4GL objects. This means that Baan (Triton) 3.1a and later can take advantage of QKEY.

Note that the QKEY 1.x versions were written in Perl (4.x and 5.x) and were held to a limited release. However, these versions were available on all Unix platforms, as the only requirement was that Perl 4.0 or later be available. If you need a copy of one of these versions they may or may not be available. This will require a special agreement between your company and QCI as these are distributed in source code form and trademark secrets can be fully viewed.

Supported Environments

Version	Operating System	Development	Runtime Library	Availability
1.11	All Unix	Yes	Yes	Controlled
1.20	All Unix	Yes	Yes	Controlled
1.21/1.22	All Unix	Yes	Yes	Special
	Windows NT	No	Yes	
2.00	HP-UX 10.x and later	Yes	Yes	General
	All other Unix	No	Yes	
	Windows NT	No	Yes	
2.10	HP-UX 10.x and later	Yes	Yes	General
	All other Unix	No	Yes	
	Windows NT	Yes	Yes	

Release History

Version 2.10

Version 2.10 is a completed port to the Windows NT environment. QKEY is now available on Windows NT and Unix systems (see the section on *Supported Environments* for an accurate list).

Version 2.00

QKEY is fully rewritten in the "C" language. This improves performance and also allows for more manageable distribution and licensing of the product.

Change this products name from ENGEN to QKEY for public release.

Version 1.22

Began porting to Windows NT environment of Baan. This version fixes some small bugs in the search path scans as Windows NT includes driver letters, which also have a colon (:). The colon was previously used only as a path separator character. This fix requires an update to both the QKEY generator and the Baan library tccomqcidll1.

Note: This version was never released.

Version 1.21

This is a fix release to adjust for some changes in Baan IVc. This version of Baan has changed on Unix to more closely match the Windows NT version. Message output had to be redirected to a different location so Baan would pick it up in the display window.

Cleaned up some warning messages that were generated by the Baan compiler when using the "optimized" call method.

Fixed a problem resulting from the use of the string "${BSE}" in the file paths for Package VRCs. The Baan IVc release defaults to using this, however, this could cause problems in prior releases as the paths are not properly searched.

Note: This version was never released.

Version 1.20

In addition to some minor bug fixes, QKEY 1.20 removes the restriction that an QKEY 4GL object must be derived directly from a standard 4GL object. With this additional capability QKEY can be used in at multiple levels in a single Package VRC structure. For example, if you are making enhancements to customer maintenance (tccom1101m000) which comes in Baan localized VRC B40L_b2_glo0 and have two VRCs derived from this, let's say B40C_b2_dev1 and B40C_b2_dev2. You can have QKEY code in both dev1 and dev2 without conflict. In both cases, QKEY will derive from the Baan localized object at the glo0 level. This added functionality requires that all previous QKEY code be recompiled as QKEY implements this feature by adding a special external function to each of your QKEY 4GL objects. By doing this the routine to load the parent DLL object can detect when it attempts to load an QKEY object and bypass it. Note: Source created with QKEY 1.1 and earlier need to be recompiled if you only use QKEY at a single level in your VRC tree.

Added "function call optimization". This is optional and still in testing. Basically function call optimization gets all the function identifiers for the parent event handling code when the program starts. Then, when the parent must be called, the QKEY 4GL object can call the parent immediately without first

having to lookup the function id in the parent. Function call optimization does add some overhead to the start up code (before.program) section so it is optionally available. Please contact a representative from QCI to learn how to activate this feature.

Fixed a bug where the "default:" tag for the case statement was considered a section name.

Added identifying marks for all QKEY messages to more clearly delineate these from the Baan standard Generator and Compilers.

Added a new utility script called *endiff* to be used to make source file comparisons. This is similar to *envi* and *enview* and will temporarily strip out QKEY added code so the base source script can be compared with another source script. In other words the script commands added by QKEY between |#pobj and |#end are removed so they don't interfere with the comparison. To learn about how to use this utility script, see the section *User Configuration.*

Version 1.10

Initial publicly available release.

Upgrading from v 1.x to v 2.x

Again, follow all instructions for *Installation*. There is again a new version of the dll library. be sure to install this as well. Use the same instructions given from *Upgrading from v 1.10 to v 1.20* below for determining all the places you may have installed a previous copy of the dll.

Upgrading from v 1.10 to v 1.20

Follow all the instructions for *Installation*. Do not leave out the final step – loading the support dll library – even though you already have a copy installed. There is a new version of the library in version 1.20. Be sure to load this library to every VRC the previous one was loaded in. A good way to verify you have gotten them all is to use the report Print Program/Library Scripts and use the sort for Program/Library then VRC. On the report, select package tc, all VRCs, and the library "comqcidll1". The report will show you all the installed VRCs. Also remember to check all your Baan environments if you have more than one.

Installation in Unix

The primary preprocessor program (the core of QKEY) is a single program which will be installed in the place of the Baan standard 4GL generator program. The 4GL generator will be renamed so it can still be used by QKEY. The installation will also install the command scripts *envi* and *endiff* which can be used to strip out QKEY generated code before using the vi and diff commands (see *User Configuration*).

1. Copy the distribution .tar (qkey-v2.00.tar or qkey-v2.00.tar.gz) file from the diskette to your host system. Use a program similar to ftp. If you are using ftp, be sure to choose "binary" transfer mode.

Example.

In this example, the name of the system where QKEY will be installed is "pluto" and the directory which will hold the installation files is called "installdir" in the bsp user's home directory.

ftp pluto
bsp
<enter password>
binary
cd installdir
put qkey-v2.00.tar or *put qkey-v2.00.tar.gz*
quit

2. Login to the host as "bsp" and be sure BSE is correctly set for your Baan environment. If not, type "BSE=<bse directory>; export BSE".

Example.

If the Baan Shell Environment is /baan/baan4/bse, you would use this.

BSE=/baan/baan4/bse
export BSE

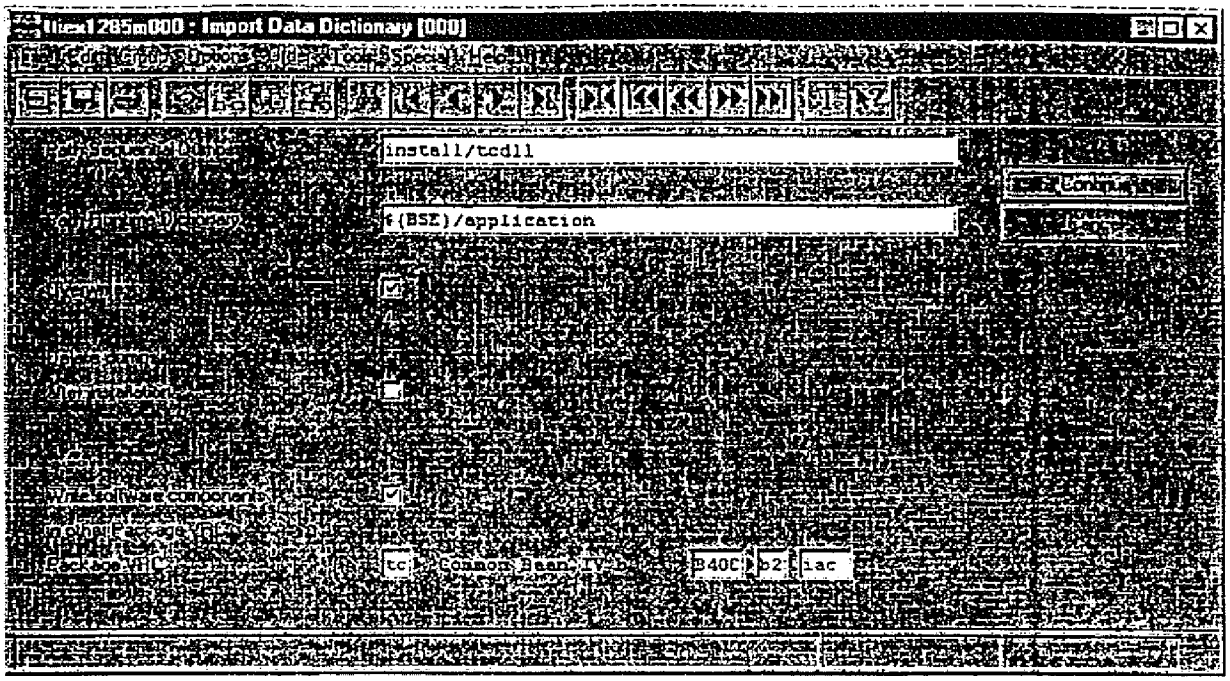
3. Change to the directory containing the QKEY installation file.

4. If you have received a compressed tar file (the file name ends with ".gz" extension) then decompress the file using "gzip -d qkey-v2.00.tar.gz". This will change the file name to "qkey-v2.00.tar".

5. Expand the .tar file by typing "tar xvof qkey-v2.00.tar"

6. Type "sh ./install.qkey". This will copy the correct qkey command file to $BSE/bin, rename std_gen6.1 and create a symbolic link to qkey. The utility edit scripts, envi, enview, and endiff will all be installed at the same time. The install.qkey script may be executed multiple times without causing problems.

7. Import the QKEY DLL into each tc package VRC where you will be using QKEY. Use Import Data Dictionary to load the library source and object; the directory is "<installpath>/tcdll" and select "load to different Package VRC" and enter your PVRC. You can load this in a single higher level VRC that all other custom VRCs derive from as it will be visible in any lower level VRC. In Baan IVa or earlier, run "patch objects after error solving" to update the object header so it passes the license check.



8. At your discretion, remove all the installation files (you may want to keep them around as they are small and if you create other VRC structures you may need to import the DLL again). Note, a Microsoft Word formatted document is also included in the distribution which contains this manual (qkey2.00.doc).

REMEMBER THIS! Any time you upgrade the porting set in Baan, you will need to reinstall QKEY. If you want, you can make the changes by hand to reactivate QKEY; as follows:

```
cd $BSE/bin
mv std_gen6.1.real std_gen6.1.real-        # Save a copy of the previous version like Baan does
mv std_gen6.1 std_gen6.1.real
ln –s qkey std_gen6.1
```

User Configuration in Unix

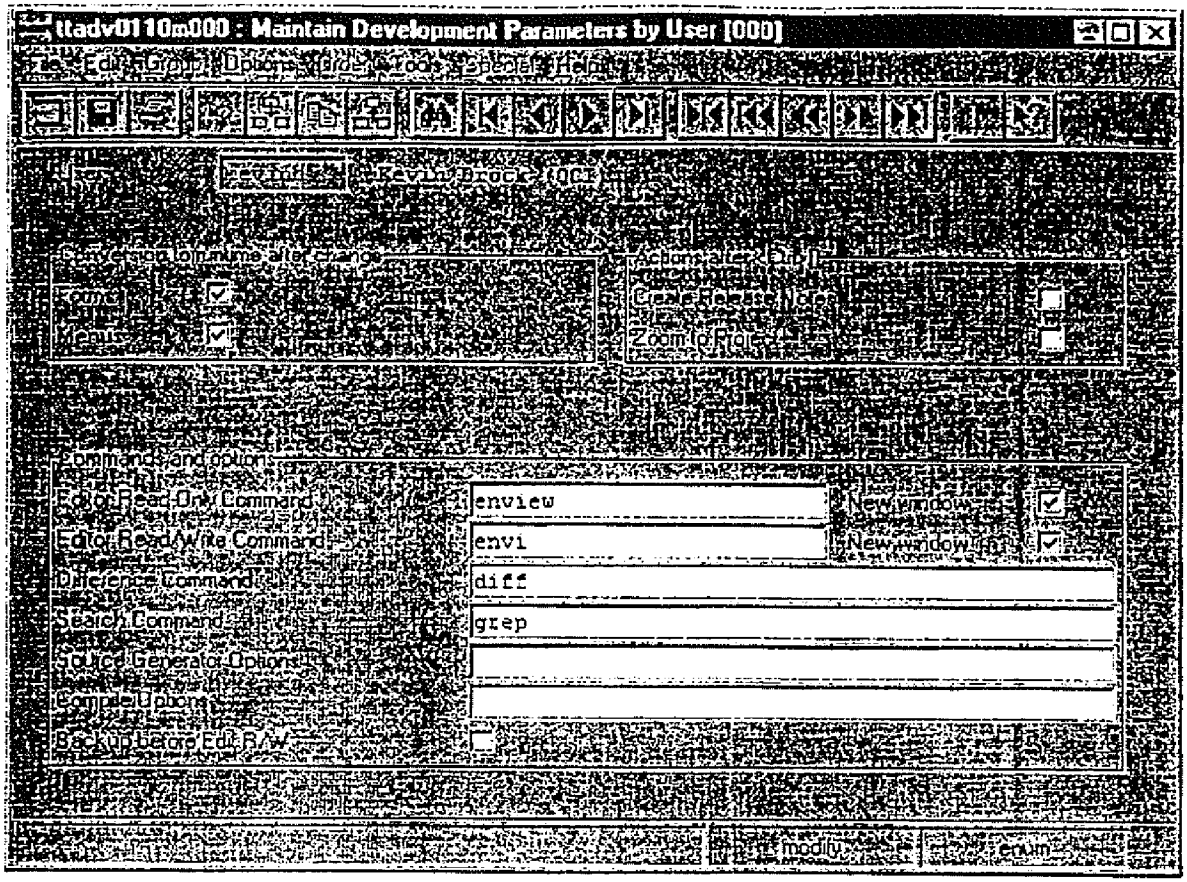
Unless you want to have your developers access the *envi* or *endiff* scripts, you will not need to change any user configuration entries in Baan.

The *envi* script is available as a preprocessor for the source code prior to starting "vi" or "view" – the standard Unix editors for Baan scripts. If you want to use a different editor, copy *envi* to a new name and change the script to suit your purposes.

This script will strip out any code between the "|#pobj" and "|#end" directives before starting vi/view. These directives delineate the code automatically created by the QKEY process. By stripping the automatic code out, you may find it easier to edit and/or view your scripts. Nothing will be done to normal (non-QKEY) scripts as they will not contain |#pobj/|#end directives.

To activate *envi* or *enview* for a user, change the Developers Data for that user. The fields to change are "Editor Read-Only Command", which should be set to "enview", and "Editor Read/Write Command" which should be set to "envi". You can chose to only set one of the commands to the en* version (e.g. envi instead of vi).

The *endiff* command is intend to help make source comparisons easier to read. The QKEY code will be stripped prior to the *diff* command being executed. This means that only the code you created is actually compared and this is the only significant code – QKEY generated code is only important for calls to the parent object to keep the entire application program functioning as expected. To use *endiff*, set the "Difference Command" field to "endiff" instead of "diff" (not shown in example).



Remember, the debugger will always show all the code between the |#pobj/|#end directives so the line numbers will still be correct unless you edit the script and don't recompile it.

Caution! Using "envi" will make it easier to edit your scripts, however, you may find it difficult to find the lines numbers reported by the Baan compiler. The compiler works with the source with all the code generated by QKEY as well as your source so line numbers will appear different to it than to the editor if **and only** if the generated lines are stripped (as with envi).

Hint. I use "envi" for my Editor Read/Write Command and "view" still for my Read Only Command. This allows me to see the source as the compiler does and get to a specific line number by selecting View from the maintenance session instead of Edit. You can make changes and save them with "view" by using the ":w!" or ":x!" commands (these override the Read Only value set in vi).

Concepts

Objects as DLLs

Unlike in previous version, in Tools 6.1 all Baan 4GL programs are really compiled into DLL objects. These objects are dynamically loaded and processed using the precompiled standard program. This is different than prior versions of Tools, where the source for the standard program was merged with the 3GL generated source for the custom 4GL script before compiling. Understanding this concept is crucial to understanding QKEY. In essence, all Baan 4GL objects are just function libraries holding the variables and code for the sections/subsections and functions of the script.

Since these objects are DLL's, the Baan standard object can be loaded dynamically as a DLL to your script. The only problem at this point is generating all the external calls at the right sections and subsections so the original functionality remains intact. The solution is QKEY, which generates all the necessary calls by retrieving all exported DLL functions that match section/subsection function names.

This opens up a number of possibilities in coding as it becomes more like traditional OOP. The "parent" object can be viewed as an OOP object, holding the inherited data (the external variables and tables) and methods (the section and subsection code). You can then add your own data and can choose to call the ancestor's methods or override (not use) those methods. For the standard program to recognize the functions in the session object, each section/subsection function must be exported by the object. Thus, all sections and subsections used by the parent object must be declared in any derived objects – QKEY transparently manages this for you. The variables (and tables) don't have to be declared in the initial object loaded by the standard program, they just need to be made visible by one of the DLLs loaded at the time they are resolved to actual addresses – sometime after the "before.program" section executes.

The Child Object

If your enhancements need to refer to tables or variables declared by the Baan standard program, they will need to be declared again (as "extern") in your script. The Baan compiler will be working only with your source and will not be checking any parent object for declarations. This works because the "extern" variables all point to the same memory location within the single bshell process.

There is some special code that must be added to the before.program section. In this section, the parent object is located and loaded. To locate the parent, there is a function that will search for the actual name of the object above the current object's location in the Package VRC tree. It does the search by first locating the object using Baan's pathname() function. Then, it opens the $BSE/lib/fd.6.1.<pacc> file and finds the PVRC search path for the object, scans the path until it passes the original object and then locates the next available one. The final object found is the one returned by the function – this gives us the object immediately "above" in the VRC search path. It has to be done this way, because to use just the object name in the load_dll() function, would cause a loop as the program tries to load itself and bshell will quickly have a stack error.

In summary, QKEY takes the steps necessary to determine what sections and subsections are used by the parent object, creates the necessary calls to these exported functions at the appropriate locations in your script, and adds code necessary to dynamic search for and load the parent object. If the new script contains a section that was previously used, QKEY adds to the code you have written (see |#inhibit and |#call for ways to control this). If there is a section that you haven't used, then the section is added automatically.

The commands for QKEY are always prefixed with "|#" which is interpreted by the Baan standard generator and compiler as comments. In this way these commands can be left in the code without confusing the compilation process. All code added by QKEY is placed between a pair of directives, |#pobj and |#end. **Do not attempt to**

place anything between these directives. Anything placed between these will be striped the next time QKEY (or envi/enview) runs so it can regenerate the new code based on the current source.

The Derivation Process

Here's a brief description of how QKEY changes the development process and execution of Baan objects. In this description we'll be working with the sales detail line object and script (session tdsls4102s000), the Baan standard code is in VRC tdB40_a and the custom code is in tdB40C_a_cust. The original object and source are physically stored as:

object: $BSE/application/tdB40_a/otdsls/osls4102
source: $BSE/application/tdB40_a/ptdsls/psls41020

Under the normal (non-QKEY) developed method you would see this:

Copy source (using copy to current PVRC from script session) to custom PVRC; file $BSE/application/tdB40C_a_cust/ptdsls/psls41020 created.

Edit source as needed.

Compile source – new object is created as $BSE/application/tdB40C_a_cust/otdsls/osls4102 and Baan no longer uses (or cares about) the original object in $BSE/application/tdB40_a.

In time, when the standard source and object are patched, the new source file will need to be changed by hand with the same fixes. This same problem will occur when moving to a new VRC tree and there is a new version of the source file.

Using the enhanced QKEY method:

Create new, empty script, in custom PVRC (td40C_a_cust). Initially this is empty. Note – The source file in $BSE/application/tdB40_a/ptdsls is not needed with QKEY.

Add to empty source as desired – only adding code that you want. |#parent must be placed in the declaration section. QKEY will add any necessary sections/sub-sections including the before.program section if you don't already have it (or add to the one you do have).

Compile source – derived object created in td40C_a_cust. When Baan runs this object, the first thing it does is search for the parent object ($BSE/application/tdB40_a/otdsls/osls4102) and load it. Later, calls will be made into the parent object as each section/sub-section is executed in the custom version of the object.

In time, when the original object is patched, at most the custom object will just need to be recompiled. However, in most circumstances, the custom object can be left alone and the patches will immediately be used.

Using QKEY

Once installed, the QKEY preprocessor is transparent. It replaces the Baan standard generator (std_gen6.1) so Baan will automatically run QKEY instead of the generator. If QKEY finds no errors and was able to successful process the source file, it will call the original generator to finish the process. Here is a list of steps you need to take to use this capability in a new session script.

1. Start a new script. If you try to copy a Baan standard script entry (and you don't have source code), the tools record will be copied, but there will still be no source file. You will not be able to edit the file as Baan requires at least an empty file to exist. In this case you will need to create an empty file in the appropriate directory ($BSE/application/....). An easier way is to call up the Baan script entry in Tools and insert a new record; accept all the defaults as they should come from the entry you just brought up. Baan will then create a new script file for you.

   Remember: If you use CTRL-X in Scripts to copy to current PVRC, the script record is copied but there's no source so that's not copied. Then when you try to use "edit source" command, there is still no source file so Baan just gives an error message. You then must create the empty file to bypass this error.

   Note: If you have Baan source and you still wish to use QKEY (there are several reasons to), remember that Baan will copy the source script to your new VRC when you choose either "copy" or "insert". There is no way around this. Just go ahead and let it make the copy, start the editor, and delete all the lines. You can then start with a clean source file and add just your enhancements.

2. Add the supplied DLL – `tccomqcidlll` – to the list of libraries for the script. (The find_parent function could have been added to each source file by QKEY but this was placed in a common DLL so the code only exists once; any fixes to find_parent can be made once).

   Note for those upgrading to Version 1.20 and later from earlier versions. The updated "find_parent" function is compatible with all previous compiled versions of QKEY and there is no need to recompile your source. However, to be able to work in 2 or more VRC levels with QKEY you still will need to recompile as the old objects don't have the identifying external function.

3. To create a derived object, add the "|#parent" directive to the declaration section of the new source code. The QKEY generator will automatically create all the code need to call the parent object and change the before.program section to load the parent object dynamically. All code added by QKEY is placed between the directive pair "|#pobj" and "|#end" (see Reference section for warnings about placing code between these directives).

   Example.
   declaration:
   table *ttiitm001*
   table *ttdsls041*

   |#parent

   If the "|#parent" command is left out, QKEY doesn't do anything to the source except strip any "|#pobj" and "|#end" pairs from the code. If no derivation directives had previously been used then the source file is left unchanged.

4. If the you do not want the code for a specific section/subsection in the parent called, the "|#inhibit" directive can be placed in that specific section/sub-section of the new source file.

   Example.
   tdsls042.oqua:
   check.input:
   ... your new code ...

   The parent object's check.input code will NOT be called for the tdsls042.oqua field.

|#inhibit

5. If you want the code for a specific section/subsection in the parent called before the end of the section/subsection, place the "|#call" directive at the point you want the parent object called.

Example.
tdsls042.oqua:
check.input:
... your new code ....
|#call → *The parent object's check.input code will be called now.*
...some more of your code....

Debugging

All code you create will be fully accessible in the Baan Tools debugger. The QKEY preprocessor leaves the generated code in your source file between the |#pobj/|#end directives so the debugger has the correct line number information. Notice that when the parent object is called, the debugger will not follow code execution into it unless it was also compiled with debugging. If you do not have the original source code you will not be able to debug into the parent object.

Reference

Following is a list of the directives understood by the QKEY preprocessor for controlling the derivation process.

|#parent [object]

This statement may appear only once in the source script and it must be in the declaration section. This activates the QKEY preprocessor on the script. Optionally, an object name can be specified so a script can derive from an object that does not match the script name (see example 2).

Example 1.
declaration:

```
... ...
|#parent
... ...
```

In this example, QKEY will seek out the parent object that matches the source file name. For instance, if the current source name is ptiitm01010 then the object name will be otiitm0101.

Example 2.
declaration:

```
... ...
|#parent otdsls4102
... ...
```

Here, QKEY will seek out object otdsls4102, no matter what the source file name is. The object found by QKEY will always be in a higher level VRC, even if there is an object in the current VRC level called otdsls4102.

|#inhibit

This directive can be used once per section (only if the section actually can have code, such as before.program) or subsection to keep the standard parent code from being executed. This will probably be used rarely as the larger use for QKEY is to add functionality to standard Baan sessions.

|#inhibit and |#call are mutually exclusive and should not be used in the same section/subsection.

Example.

```
field.tdsls042.oqua:
when.field.changes:
    ... ...
    tot.wght = tot.wght - prev.wght + (tdsls042.oqua * tiitm001.wght)
    ... ...
    |#inhibit
```

Here, the parent code for field tdsls4102.oqua, subsection when.field.changes will not be called. All other sections and subsections are uneffected. In this example, |#inhibit cannot be used immediately following the "field.tdsls4102.oqua" section name as code can not be legal placed at this point.

|#call

This directive can be used once per section or subsection to specify when the parent object code is called. Normally the parent object is called at the end of the section/subsection so any new code you add will be executed first.

|#inhibit and |#call are mutually exclusive and should not be used in the same section/subsection.

Example.

on.main.table:
before.delete:
... delete unrelated new rows here ...
|#call
... delete additional related rows here ...

The parent object routine for before.delete will be called after executing the first set of lines and before the second set of lines.

|#pobj

Delineates the start of code automatically added by QKEY. **Do not place code after this directive and before the succeeding |#end directive.** Code placed between |#pobj and |#end will be removed every time QKEY runs to strip what was generated on the previous run.

Caution! Do not add or remove |#pobj statements. It is best to use the QKEY (or envi) preprocessor to manipulate these. To remove all the directive pairs you can simply remove the |#parent directive from your script and recompile it.

Example.

```
|#parent
|#pobj added by QKEY 2.00
long   _pobj_dll_id
long   _pobj_func_id
long   _pobj_err
string _pobj_path(255)
|#end add by QKEY
```

|#end

Delineates the end of code automatically added by QKEY. **Do not place code before this directive and after the preceding |#pobj directive.** Code placed between |#pobj and |#end will be removed every time QKEY runs to strip what was generated on the previous run.

Caution! Do not add or remove |#end statements. It is best to use the QKEY (or envi) preprocessor to manipulate these. To remove all the directive pairs you can simply remove the |#parent directiven from your script and recompile it.

(see |#pobj for example)

Benefits

1. The new script only needs to contain the enhancements to the original code.

2. The new object dynamically loads the parent, thus, if the parent is changed the updated code is executed by the child. This could be helpful when upgrading to a new version of Baan. The way the program is created with QKEY, the custom object does not even need to be recompiled as long as the parent object has the same sections and subsections.

3. In the case that the sections and subsections in the parent have changed, the derived object only needs to be recompiled (QKEY takes care of adding the necessary code to the child script).

4. You need not have source code to create complex customizations to a standard session.

Limitations

There are some limitations with QKEY, but these limitations do not eliminate any current Baan functionality.

1. QKEY cannot change any current standard code (3GL functions). You can only add code around standard Baan routines or choose to not execute a standard routine. For example, in a when.field.changes subsection, which is part of a field section, you can choose to execute code before or after the Baan code for when.field.changes (using |#call). You can also choose to not execute Baan's standard code for this subsection (using |#inhibit), however, you **cannot** change what happens in Baan's standard code for this subsection.

2. QKEY cannot be used with 3GL code or report objects as there are few sections (none in 3GL code) that you can take advantage of. It has been designed to work with code tied to forms. In addition, std_gen6.1 is only executed for 4GL scripts; this is the only time QKEY will be called as it replaces the std_gen6.1 binary.

3. Limitation 3 removed in QKEY version 1.20! QKEY will automatically search for the real parent object in the VRC path. Any objects with the same name and generated by QKEY will be bypassed automatically in this search.

Examples

Here is some sample code for changes made to the sales order detail line display session (tdsls4503s000). Notice that QKEY was used to add calculations and new variables for the form.

Example 1.

Here is the code as written by the programmer.

```
|*****************************************************************************
|* tdsls4503  0  VRC B40C b2 iac
|* Display line items during order entry
|* bsp
|* 05-22-97 [15:11]
|*****************************************************************************
|* Script Type: 123
|*****************************************************************************
|****************************** DECLARATION SECTION ***************************
declaration:
       table  tdltc001     | Lot information
       table  tdsls041     | Sales order detail

       extern domain tdltc.clot    sls.lot
       extern domain tcamnt        l.neta  | Net price for line

       |#parent

|****************************** PROGRAM      SECTION ***************************
|****************************** ZOOM FROM    SECTION ***************************
|****************************** FORM         SECTION ***************************
|****************************** CHOICE       SECTION ***************************
|****************************** FIELD        SECTION ***************************
field.sls.lot:
before.display:
       | Run number is the first 4 characters from the lot (if specified)
       if tdsls041.lsel = tclsel.any then
              sls.lot = ""
              tdltc001.inf1 = ""
       else
              sls.lot = shift1$(tdsls041.clot)
              | Get the lot record for the selection code information
              select tdltc001.*
              from   tdltc001
              where  tdltc001._index1 = { :tdsls041.cprj,
                            :tdsls041.item, :tdsls041.cntr, :tdsls041.clot }
              selectdo
                    continue
              endselect
       endif

field.l.neta:
before.display:
       | Compute net amount for item (Extended / Qty Ordered)
       l.neta = tdsls041.amta / tdsls041.oqua

|****************************** MAIN TABLE   SECTION ***************************
|****************************** FUNCTION     SECTION ***************************
```

Example 2.

Here is the same code as shown in Example 1 after it has been processed by QKEY version 1.10 and compiled by Baan. QKEY 1.20 and later are similar though they use some defines and have the potential of creating a table of functions identifiers for faster execution. Function optimization is still in testing so may or may not be seen in your generated source.

```
|*****************************************************************************
|* tdsls4503  0  VRC B40C b2 iac
|* Display line items during order entry
|* bsp
|* 05-22-97 [15:11]
|*****************************************************************************
|* Script Type: 123
|*****************************************************************************
|****************************** DECLARATION SECTION ***************************
declaration:
       table  ttdltc001     | Lot information
       table  ttdsls041     | Sales order detail

       extern domain tdltc.clot   sls.lot
       extern domain tcamnt       l.neta   | Net price for line

       |#parent
       |#pobj  --ADDED BY QKEY
       long   _pobj_dll_id
       long   _pobj_func_id
       long   _pobj_err
       string _pobj_path(255)
       |#end  --ADDED BY QKEY

|****************************** PROGRAM      SECTION ***************************
|****************************** ZOOM FROM    SECTION ***************************
|****************************** FORM          SECTION ***************************
|****************************** CHOICE       SECTION ***************************
|****************************** FIELD        SECTION ***************************
field.sls.lot:
before.display:
       | Run number is the first 4 characters from the lot (if specified)
       if tdsls041.lsel = tclsel.any then
            sls.lot = ""
            tdltc001.infl = ""
       else
            sls.lot = shiftl$(tdsls041.clot)
            | Get the lot record for the selection code information
            select tdltc001.*
            from   tdltc001
            where  tdltc001._index1 = { :tdsls041.cprj,
                        :tdsls041.item, :tdsls041.cntr, :tdsls041.clot }
            selectdo
                  continue
            endselect
       endif

field.l.neta:
before.display:
       | Compute net amount for item (Extended / Qty Ordered)
       l.neta = tdsls041.amta / tdsls041.oqua
       |#pobj  --ADDED BY QKEY

|*****************************************************************************
|* Autogenerated sections/subsections for all remaining exported functions
|* from parent object.
|*****************************************************************************
```

```
before.program:
        | This must be executed first; it loads the parent object
        | dll so it's routines can be called.
        | The "find_parent_obj" is an exported function that is in
        | the tccomqcidlll library.
        if (find_parent_obj("otdsls4503", "otdsls4503", _pobj_path) < 0) then
          message("Unable to find parent object dll!")
          stop()
        endif

        _pobj_dll_id = load_dll(_pobj_path,0)
        if (_pobj_dll_id <= 0) then
          message("Unable to load parent object dll!")
          stop()
        endif
        _pobj_func_id = get_function(_pobj_dll_id, "before.program")
        _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

field.itm.dsca:
before.display:
        _pobj_func_id = get_function(_pobj_dll_id, "before.display.itm.dsca")
        _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

field.tdsls041.disc:
before.display:
        _pobj_func_id = get_function(_pobj_dll_id,
                        "before.display.tdsls041.disc")
        _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

field.tdsls041.pono:
before.display:
        _pobj_func_id = get_function(_pobj_dll_id,
                        "before.display.tdsls041.pono")
        _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

form.1:
init.form:
        _pobj_func_id = get_function(_pobj_dll_id, "init.form.1")
        _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

        |#end  --ADDED BY QKEY

|****************************** MAIN TABLE  SECTION ****************************
|****************************** FUNCTION    SECTION ****************************
```

```
/*****************************************************************************
  QSET (previously known as ENGEN) for Baan Tools 6.1 and later

  Author     : Kevin Brock
  Company    : Quality Consultants, Inc. (Process Technology Group)
  Created    : May, 1997 & March, 1998
  Tools Vers: 6.1

  Copyright (C) 1997,1998 Quality Consultants, Inc.
=============================================================================
Version     Date          Who  Comments
------- -------  ------ ---------------------------------------------------
        05/17/97 KRB    Initial version started
  2.0   03/28/98 KRB    Port ENGEN from Perl to C
*****************************************************************************/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pwd.h>
#include <ctype.h>

#include "misc.h"

#define COPYRIGHT "Copyright (C) 1997,1998 Quality Consultants, Inc."
#define VERSION   "2.0"
#define MYNAME    "ENGEN"

#define BAANTOOLSVER "6.1"

#define FILENAMESIZE    30
#define PATHSIZE        256
#define OBJNAMESIZE     100

#define EOS             '\0'
#define WHITESPACE      " \t\n\r"
#define nexttok(x)      (strtok(NULL, (x)))
#define strpartcmp(x,y) strncmp((x), (y), strlen(y))

struct pathst {
  char *pathcode;
  char *path;
  struct pathst *next;
  };

#define S_SECTION       1
#define S_SUBSECTION    2

struct sectionst {
  char *name;
  char *extfunc;        /* Exported function name */
  int  output_done;     /* =1 when output by |#call */
                        /* =2 when output at end of section/subsection */
  int  stype;           /* =1 for sections */
                        /* =2 for subsection */
  struct sectionst *firstsub;
```

```
  struct sectionst *next;
  struct sectionst *secp; /* This points back to the section structure */
                          /* Note: For sections, this points to itself */
  };

struct commentst {
  char *comment;
  struct commentst *next;
  };

/* Parameter settings */
int  optimize_function_calls = FALSE;

/* Environment */
char bse[PATHSIZE];
char bsetmp[PATHSIZE];
char user[20];
char pacc[20];
char bic_info[FILENAMESIZE];
char std_gen[FILENAMESIZE];

/* Internal global variables */
int  emit_holder = TRUE;  /* Time to output "|#pobj" directive       */
int  inhibit     = FALSE; /* Inhibit directive */
int  bp_found    = FALSE; /* before.program found in normal processing */
char current_section[OBJNAMESIZE];
char current_subsection[OBJNAMESIZE];
int  lineno        = 0;
int  numfuncs    = 0;
int  genobj      = 0;
int  lastlvl       = -1;   /* lastlvl is set by search_path and is the count
                   of the levels searched when looking through
                   the passed path name.  It would be used to
                   skip the same number of directory entries in
                   the path or similar path; generally to find
                   something further up the search path (parent
                   objects, etc). */

char command[MAXLINE]; /* temporary buffer; should be used quickly */

FILE *destfile;
FILE *srcfile;
char src_name[FILENAMESIZE];
char srcfullpath[PATHSIZE];
char src_line[MAXLINE];
char src_package[3], src_module[4], src_number[5];
char current_obj[FILENAMESIZE];
char parent_obj[FILENAMESIZE];

int save_stdout;
int save_stderr;
char baan4c_output[PATHSIZE];
char temp_output[PATHSIZE];

struct sectionst *s_list = NULL;  /* List of sections/subsections in parent */
struct sectionst *s_prev = NULL;  /* The previous section when searching */
struct sectionst *sb_prev = NULL; /* The previous subsection when searching */
```

```
struct pathst *path_list = NULL;  /* List of paths */
struct pathst *path_last = NULL;  /* End of path list */

struct commentst *cmt_list = NULL; /* List of comments */
struct commentst *cmt_last = NULL; /* End of comment list */

/* Prototypes as needed */
void end_program(int exitcode);

/*****************************************************************************
 * Support routines                                                          *
 *****************************************************************************/

void chop(char *s)
  {
  int l = strlen(s);

  if (l > 0)
    *(s + l - 1) = EOS;
  }

void rename_file(char *oldname, char *newname)
  {
  if (link(oldname, newname) == 0)
    {
    unlink(oldname);
    }
  }

static void engen_msg(char *msglevel, char *fmt, va_list ap)
{
  int   errno_save;
  char  buf[MAXLINE];
  char  fmtbuf[MAXLINE];

  errno_save = errno;
  if (msglevel == NULL)
    sprintf(fmtbuf, "%s: %s", MYNAME, fmt);
  else
    if (lineno)
      sprintf(fmtbuf, "%s (%d): %s: %s", MYNAME, lineno, msglevel, fmt);
    else
      sprintf(fmtbuf, "%s: %s: %s", MYNAME, msglevel, fmt);
  vsprintf(buf, fmtbuf, ap);
  strcat(buf, "\n");
  fputs(buf, stdout);
  return;
}

void fatal_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg("Fatal", fmt, ap);
  va_end(ap);
```

```
  end_program(1);
}

void error_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg("Error", fmt, ap);
  va_end(ap);
  return;
}

void warning_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg("Warning", fmt, ap);
  va_end(ap);
  return;
}

void info_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg(NULL, fmt, ap);
  va_end(ap);
  return;
}

/*****************************************************************************
 * Section/Subsection list handling                                          *
 *****************************************************************************/

/* Note: These lists are kept in sorted order so output is sorted */

/* Look for the section name in the section link listed */
struct sectionst *find_section(char *section, int exact)
  {
    struct sectionst *s;
    int    cmp;

    s_prev = NULL;

    /* No list established yet! */
    if (s_list == NULL)
      return (NULL);

    for (s = s_list; s != NULL && (cmp = strcmp(s->name, section)) < 0;
         s_prev = s, s = s->next);
    if (exact)
      return (cmp == 0 ? s : NULL);

    return (s);
```

```
  }

struct sectionst *find_subsection(struct sectionst *sectionp,
                                  char *subsection,
                                  int  exact)
  {
    struct sectionst *sb;
    int    cmp;

    sb_prev = NULL;

    /* No section or subsection list */
    if (sectionp == NULL || sectionp->firstsub == NULL)
      return (NULL);

    /* Subsection name is null -- there will be none then */
    if (subsection[0] == EOS)
      return (NULL);

    for (sb = sectionp->firstsub;
         sb != NULL && (cmp = strcmp(sb->name, subsection)) < 0;
         sb_prev = sb, sb = sb->next);
    if (exact)
      return (cmp == 0 ? sb : NULL);

    return (sb);
  }

int find_reference(struct sectionst **rets, char *section, char *subsection)
  {
  int ok;
  struct sectionst *s;

  s = find_section(section, TRUE);
  if (s != NULL)
    {
    if (*subsection != EOS)
      {
      s = find_subsection(s, subsection, TRUE);
      }
    else
      {
      /* If a subsection name is not given but there a subsection list then
       this entry is an error as a subsection must be specified */
      if (s->firstsub != NULL)
      s = NULL;
      }
    }

  *rets = s;
  return (s != NULL);
  }

/* Add a section and subsection to the parent's section table */
void add_reference(char *section, char *subsection, char *object)
  {
    struct sectionst *s, *sb;
```

```
/* Locate the section and subsection within the list; the
   entry may only exist one time */
s = find_section(section, FALSE);
if (s != NULL && strcmp(s->name, section) == 0)
  sb = find_subsection(s, subsection, FALSE);
else
  s = sb = sb_prev = NULL;

/* Build a new section structure */
if (s == NULL)
  {
  if ((s = malloc(sizeof(struct sectionst))) == NULL)
    {
    fatal_msg("unable to allocate memory for section %s", section);
    return;
    }
  memset(s, 0, sizeof(struct sectionst));
  s->name = strdup(section);
  s->stype = S_SECTION;
  s->secp = s;

  /* If there's no subsection name then the object name goes with
     the section */
  if (*subsection == EOS)
    {
    s->extfunc = strdup(object);
    }

  if (s_prev == NULL)
    {
    /* Insert at beginning of list */
    s->next = s_list;
    s_list = s;
    }
  else
    {
    /* Add to current position in list */
    s->next = s_prev->next;
    s_prev->next = s;
    }
  }

/* Build a new subsection structure (if there is a subsection) */
if (*subsection != EOS &&
    (sb == NULL || strcmp(sb->name, subsection) != 0))
  {
  if ((sb = malloc(sizeof(struct sectionst))) == NULL)
    {
    fatal_msg("unable to allocate memory for subsection %s:%s", section,
          subsection);
    return;
    }
  memset(sb, 0, sizeof(struct sectionst));
  sb->name = strdup(subsection);
  sb->extfunc = strdup(object);
  sb->stype = S_SUBSECTION;
```

```
        sb->secp = s;

        if (sb_prev == NULL)
          {
          /* Insert at beginning of subsection list */
          sb->next = s->firstsub;
          s->firstsub = sb;
          }
        else
          {
          /* Add to current position in subsection list */
          sb->next = sb_prev->next;
          sb_prev->next = sb;
          }
        }
    }

void dispose_sections(void)
  {
  struct sectionst *s, *sb, *stemp;

  /* Free the memory for the sections/subsections list */
  for (s = s_list; s != NULL;)
    {
    for (sb = s->firstsub; sb != NULL;)
      {
      if (sb->name != NULL)
        free(sb->name);
      if (sb->extfunc != NULL)
        free(sb->extfunc);
      stemp = sb;
      sb = sb->next;
      free(stemp);
      }

    if (s->name != NULL)
      free(s->name);
    if (s->extfunc != NULL)
      free(s->extfunc);
    stemp = s;
    s = s->next;
    free(stemp);
    }

  s_list = NULL;
  s_prev = NULL;
  sb_prev = NULL;
  }

/* Call this to begin enuermating all sections and subsections */
/* Note: Only used entities are returned.  In other words, a   */
/*   section that has subsections is not returned by itself.   */
struct sectionst *enum_s;
struct sectionst *enum_sb;

struct sectionst *enum_sections(void)
  {
```

```
  enum_s = s_list;
  enum_sb = enum_s->firstsub;
  return (enum_sb == NULL ? enum_s : enum_sb);
  }

/* Call this to get next section or subsection in enumerate list */
struct sectionst *enum_next(void)
  {
  if (enum_sb != NULL)
    {
    enum_sb = enum_sb->next;
    if (enum_sb != NULL)
      return (enum_sb);
    }

  /* Move to next section */
  enum_s = enum_s->next;
  enum_sb = enum_s->firstsub;
  return (enum_sb == NULL ? enum_s : enum_sb);
  }

/*****************************************************************************
 * Comment handling                                                          *
 *****************************************************************************/

void add_comment(char *comment)
  {
  struct commentst *c;

  c = malloc(sizeof(struct commentst));
  c->comment = strdup(comment);
  c->next = NULL;

  if (cmt_list == NULL)
    {
    cmt_list = c;
    cmt_last = c;
    }
  else
    {
    cmt_last->next = c;
    cmt_last = c;
    }
  }

void dispose_comments(void)
  {
  struct commentst *c, *ctemp;

  for (c = cmt_list; c != NULL;)
    {
    free(c->comment);
    ctemp = c;
    c = c->next;
    free(ctemp);
    }
```

```
  cmt_list = NULL;
  cmt_last = NULL;
  }

/*****************************************************************************
 * Pathlist handling                                                         *
 *****************************************************************************/

void add_path(char *pathcode, char *path)
  {
  struct pathst *p;

  p = malloc(sizeof(struct pathst));
  p->pathcode = strdup(pathcode);
  p->path = strdup(path);
  p->next = NULL;

  if (path_list == NULL)
    {
    path_list = p;
    path_last = p;
    }
  else
    {
    path_last->next = p;
    path_last = p;
    }
  }

struct pathst *find_path(char *pathcode)
  {
  struct pathst *p;

  for (p = path_list;
       p != NULL && strcmp(p->pathcode, pathcode) != 0;
       p = p->next);

  return (p);
  }

void dispose_paths(void)
  {
  struct pathst *p, *ptemp;

  for (p = path_list; p != NULL;)
    {
    free(p->pathcode);
    free(p->path);
    ptemp = p;
    p = p->next;
    free(ptemp);
    }

  path_list = NULL;
  path_last = NULL;
  }
```

```
/*****************************************************************************
 * Object search and load routines                                           *
 *****************************************************************************/

/* is_obj_engen() function checks to see if the obj file an engen
   generated object (it includes the function "_engen_object_version") */
int is_obj_engen(char * filename)
  {
  FILE  *pfile;
  int   is_engen;
  char  linebuf[MAXLINE];

  is_engen = 0;

  /* Open a pipe from bic_info (Baan's tool to report information on
     objects) */
#ifdef WINNT
#else
  sprintf(command, "%s/bin/%s -e %s", bse, bic_info, filename);
  if ((pfile = popen(command, "r")) == NULL)
    error_msg("unable to start %s; aborting", bic_info);
#endif

  while (fgets(linebuf, MAXLINE, pfile) != NULL)
    {
    chop(linebuf);

    if (strstr(linebuf, "ERROR") != NULL)
      {
      error_msg("%s reported an error; aborting", bic_info);
      break;
      }

    if (strpartcmp(linebuf, "function extern ") == 0 &&
        strstr(linebuf, "_engen_object_version") != NULL)
      {
      info_msg("Skip %s obj: %s", locasestr(MYNAME), filename);
      is_engen = 1;
      break;
      }
    }

  fclose(pfile);

#ifdef WINNT
#endif

  return (is_engen);
  }

/* This searches the appropriate path as retrieved from the fd6.1.<pacc>
   file.  The first parameter is the Baan standard name (ptdsls4102).
   If skiplvl is non-zero then start searching just past that position
   in the path list (skip all prior entries).

   If skipengen is true then skip any engen objects (those containing
   the extern function "_engen_object_version").  Note: This will only
```

```
   occur if skiplvl is also set.

   Returns a pointer to the full path to the file (including the file
   name) -- null if the files was not found. */
char * search_path (char * baanname, int skiplvl, int skipengen)
  {
  static char buf[PATHSIZE];
  char       filename[FILENAMESIZE], code[2], package[3], module[4], rest[21];
  char       pathcode[4], path[PATHSIZE], *p, *dir, *s, *d;
  struct pathst *pp;

  skipengen = (skiplvl ? skipengen : 0);

  /* Split the Baan standard name into the proper directory/filename */
  strcpy(code, substr(baanname, 1, 1));
  strcpy(package, substr(baanname, 2, 3));
  strcpy(module, substr(baanname, 4, 6));
  strcpy(rest, substr(baanname, 7, 26));
  sprintf(filename, "/%s%s%s/%s%s%s", code, package, module,
        code, module, rest);

  /* Locate the path name to use in the loaded path name list */
  strcpy(pathcode, code);
  strcat(pathcode, package);
  if ((pp = find_path(pathcode)) == NULL)
    return (NULL); /* No valid path found! */
  strcpy(path, pp->path);

  /* Skip past directories if requested */
  p = path;
  lastlvl = 0;
  while (skiplvl > 0)
    {
    /* To support Windows NT path names, we must skip past at least the
       second character -- we can assume the path will be at least larger
       than 2 characters and the second one may be a legitimate ":" to
       identified the drive */
    if (*(p+1) == ':') p += 2;
    while (*p != EOS && *p != ':') p++;
    skiplvl --;
    lastlvl ++;
    }

  /* Look for file in path list */
  while (p != NULL)
    {
    lastlvl ++;
    dir = p;
    /* Must skip past drive letter under Windows NT (see comment above) */
    if (*(p+1) == ':') p += 2;
    p = strstr(p, ":");
    if (p != NULL)
      {
      *p = EOS;
      p++;
      }
```

```
    /* Expand ${BSE} to the actual contents of the bse */
    if (strstr(dir, "${BSE}") != NULL)
      {
      strcpy(buf, "");
      for (s = dir, d = buf; *s != EOS;)
        {
        /* In theory this should always be at the beginning of the string,
           this can handle other situations, however, we will always
           assume that there is only one ${BSE} literal */
        if (strpartcmp(s, "${BSE}") == 0)
          {
          strcat(d, bse);
          s += 6;
          strcat(d, s);
          break;
          }
        else
          {
          *d = *s;
          d++;
          s++;
          }
        }
      }
    else
      strcpy(buf, dir);

    strcat(buf, filename);
    if (access(buf, R_OK) == 0)
      {
      if (!skipengen || (skipengen && !is_obj_engen(buf)))
        return(buf);
      }
    }

  /* No file found! */
  return (NULL);
  } /* search_path */

int loadparent(char *pobj)
  {
  char      line[256];
  char  object[100], *t, *p, parts[5][25];
  char  key1[50], key2[50];
  FILE      *dllfile;
  int   ok = TRUE;

#ifdef WINNT
#else
  /* Open bic_info6.1 command on the parent object */
  sprintf(command, "%s/bin/%s -e %s 2>&1", bse, bic_info, pobj);
  if ((dllfile = popen(command, "r")) == NULL)
    {
    error_msg("unable to start %s; aborting", bic_info);
    return (FALSE);
    }
#endif
```

```
numfuncs = 0;

while (fgets(line, 256, dllfile) != NULL)
  {
  chop(line);

  if (strstr(line, "ERROR") != NULL)
    {
    error_msg("%s reported an error; aborting", bic_info);
    ok = FALSE;
    break;
    }

  if (strpartcmp(line, "function extern ") == 0)
    {
    /* Skip past first two tokens -- that is "function" and "extern" */
    t = strtok(line, WHITESPACE);
    t = nexttok(WHITESPACE);

    /* This is the name of the external function */
    strcpy(object, (t = nexttok(WHITESPACE "(")));

    /* Is the object name one we are interested in?  It must consist only
       of Upper/Lower/Numeric characters */
    for (p = t; *p != EOS && isalnum(*p); p++);
    if (*p != '.') continue;  /* name potion must end in period */

    /* Split the object name into components based on "." delimiter */
    /* This uses the original line buffer as it's no longer needed and
       keeps the "object" variable intact for later processing. */
    strcpy(parts[0], strtok(t, "."));
    strcpy(parts[1], nexttok("."));
    strcpy(parts[2], nexttok("."));
    strcpy(parts[3], nexttok("."));
    strcpy(parts[4], nexttok("\n"));  /* Remainder of object name */

    if (strcmp(parts[1], "choice") == 0 || strcmp(parts[1], "form") == 0)
      {
    sprintf(key1, "%s.%s.%s.%s", parts[1], parts[2],
              parts[3], parts[4]);
    sprintf(key2, "%s.%s", parts[0], parts[1]);
      }
    else if (strpartcmp(object, "init.field.") == 0 ||
          strpartcmp(object, "before.field.") == 0 ||
          strpartcmp(object, "after.field.") == 0 ||
          strpartcmp(object, "before.input.") == 0 ||
          strpartcmp(object, "after.input.") == 0 ||
          strpartcmp(object, "before.display.") == 0 ||
          strpartcmp(object, "after.display.") == 0 ||
          strpartcmp(object, "before.zoom.") == 0 ||
          strpartcmp(object, "after.zoom.") == 0 ||
          strpartcmp(object, "before.checks.") == 0 ||
          strpartcmp(object, "domain.error.") == 0 ||
          strpartcmp(object, "ref.input.") == 0 ||
          strpartcmp(object, "ref.display.") == 0 ||
          strpartcmp(object, "check.input.") == 0 ||
```

```
      strpartcmp(object, "on.input.") == 0)
  {
sprintf(key1, "field.%s.%s.%s", parts[2], parts[3], parts[4]);
sprintf(key2, "%s.%s", parts[0], parts[1]);
  }
else if (strcmp(parts[2], "zoom") == 0 && strcmp(parts[3], "from") == 0)
  {
  strcpy(key1, "zoom.from.");
  strcat(key1, parts[4]);
sprintf(key2, "%s.%s", parts[0], parts[1]);
  }
else if (strpartcmp(object, "when.field.changes") == 0)
  {
sprintf(key1, "field.%s.%s", parts[3], parts[4]);
  strcpy(key2, "when.field.changes");
  }
else if (strcmp(object, "after.update.db.commit") == 0 ||
    strcmp(object, "before.program") == 0 ||
    strcmp(object, "after.program") == 0 ||
    strcmp(object, "on.error") == 0)
{
strcpy(key1, object);
strcpy(key2, "");
}
else if (strpartcmp(object, "before.read") == 0 ||
      strpartcmp(object, "after.read") == 0 ||
      strpartcmp(object, "before.write") == 0 ||
      strpartcmp(object, "after.write") == 0 ||
      strpartcmp(object, "after.skip.write") == 0 ||
      strpartcmp(object, "before.rewrite") == 0 ||
      strpartcmp(object, "after.rewrite") == 0 ||
      strpartcmp(object, "after.skip.rewrite") == 0 ||
      strpartcmp(object, "before.delete") == 0 ||
      strpartcmp(object, "after.delete") == 0 ||
      strpartcmp(object, "after.skip.delete") == 0 ||
      strpartcmp(object, "read.view") == 0)
  {
  strcpy(key1, "main.table.io");
  strcpy(key2, object);
  }
else
  {
  /* Unknown exported function format */
  warning_msg("unknown exported function: %s", object);
  strcpy(object, ""); /* Invalidate the object as we don't know how
                         to handle it. */
  }

if (strcmp(object, "") != 0)
  {
  /* Strip any unnecessary periods from the end of the key strings */
  for (p = key1 + strlen(key1) - 1; *p == '.'; p--);
  p++; *p = EOS;
  for (p = key2 + strlen(key2) - 1; *p == '.'; p--);
  p++; *p = EOS;

  add_reference(key1, key2, object);
```

```
          numfuncs ++;
          }
        }
      }

  pclose(dllfile);

#ifdef WINNT
#endif

  return (ok);
  } /* loadparent */

/*****************************************************************************
 * Emitter section                                                           *
 *****************************************************************************/

void out_startpobj(void)
  {
  if (emit_holder)
    {
    fprintf(destfile, "\t|#pobj added by %s %s\n", MYNAME, VERSION);
    emit_holder = FALSE;
    }
  }

void out_endpobj(void)
  {
  if (! emit_holder)
    {
    fprintf(destfile, "\t|#end add by %s\n", MYNAME);
    emit_holder = TRUE;
    }
  }

char *get_defname(char *extfunc)
  {
  static char nbuf[80], *p;

  /* Map a function name with "."s to a #define name.  Also prepends
     "_f_" to the name */
  strcpy(nbuf, "_f_");
  strcat(nbuf, extfunc);
  for (p = nbuf; *p != EOS; p++) if (*p == '.') *p = '_';
  return (nbuf);
  }

void out_functiondefs(void)
  {
  struct sectionst *s;
  int    cnt;
  char   def[80];

  /* This should only be called if we're optimizing the function calls   */
  /* and in the declaration section.  It creates the definitions for the */
  /* array holding the extern function ids in the parent dll for use     */
  /* later in making the direct calls.                                   */
```

```
  if (! optimize_function_calls) return;

  out_startpobj();

  fprintf(destfile, "\tlong\t_pobj_func_ids(%d)\n", numfuncs);

  for (cnt = 0, s = enum_sections(); s != NULL; s = enum_next())
    {
    cnt++;
    strcpy(def, get_defname(s->extfunc));
    fprintf(destfile, "\t#define %s%s_pobj_func_ids(%d)\n", def,
            strrep('\t', 5 - (strlen(def)) / 8), cnt);
    }
  }

void out_dllload(void)
  {
  bp_found = 1;

  out_startpobj();

  fprintf(destfile, "\t| This must be executed first; it "
                    "loads the parent object\n");
  fprintf(destfile, "\t| dll so it's routines can be called.\n");
  fprintf(destfile, "\t| \"find_parent_obj\" is an exported function "
                    "that is in\n");
  fprintf(destfile, "\t| the tccomqcidll1 library.\n");
  fprintf(destfile, "\tif (find_parent_obj(\"%s\", \"%s\", _pobj_path) "
                    "< 0) then\n", current_obj, parent_obj);
  fprintf(destfile, "\t  message(\"Unable to find parent object dll!\")\n");
  fprintf(destfile, "\t  end()\n");
  fprintf(destfile, "\tendif\n\n");
  fprintf(destfile, "\t_pobj_dll_id = load_dll(_pobj_path,0)\n");
  fprintf(destfile, "\tif (_pobj_dll_id <= 0) then\n");
  fprintf(destfile, "\t  message(\"Unable to load parent object dll!\")\n");
  fprintf(destfile, "\t  end()\n");
  fprintf(destfile, "\tendif\n");

  if (optimize_function_calls)
    {
    fprintf(destfile, "\n\t_pobj_get_fids()\n");
    }
  }

/* Output a parent call for a specific section/subsection. */
/* Use this if you have the pointer to the memory structure */
/* Returns 1 if succesful, 0 on failure */
int out_pcall_ref(struct sectionst *s, int callverb)
  {
  /* Have we already output the call information for this entity? */
  if (s->output_done)
    {
    if (callverb)
      {
      /* The "|#call" directive was specified, but we've already output */
      /* the call once, so it probably appears twice. */
```

```
      error_msg("|#call used more than once in %s:%s\n",
            s->secp->name, (s->stype == S_SUBSECTION ? s->name : ""));
      return (0);
      }

    if (s->output_done == 2)
      {
      error_msg("section %s:%s specified multiple times",
            s->secp->name, (s->stype == S_SUBSECTION ? s->name : ""));
      return (0);
      }

    /* Ignore this default generate request, the call to the parent */
    /* object was already generated with a |#call directive.        */
    s->output_done = 2;
    return (1);
    }

  if (callverb)
    s->output_done = 1;  /* Ok to see a default generate request */
  else
    s->output_done = 2;  /* Should never see this request again */

  /* Now, generate the code...unless the program has strictly inhibited this */
  if (! inhibit)
    {
    out_startpobj();

    if (optimize_function_calls)
      fprintf(destfile, "\t_pobj_exe0(%s)\n", get_defname(s->extfunc));
    else
      fprintf(destfile, "\t_pobj_exe1(\"%s\")\n", s->extfunc);
    }

  return (1);
  }

/* Output a parent call for a specific section/subsection. */
/* Use this if you have just the section/subsection names */
/* Returns 1 if succesful, 0 on failure */
int out_pcall(char *section, char *subsection, int callverb)
  {
  struct sectionst *s;

  /* Determine if the section/subsection has a parent related object */
  if (! find_reference(&s, section, subsection))
    {
    /* Parent didn't have a reference to this, consider this successful */
    return (1);
    }

  return (out_pcall_ref(s, callverb));
  }

/* Write out all inheritance calls for all remaining subsections for the */
/* passed section (these subsections are not defined in the current      */
/* source).                                                              */
```

```
void out_rest(char *section)
  {
  struct sectionst *s, *sb;
  int     first = TRUE;

  s = find_section(section, TRUE);
  if (s == NULL || s->firstsub == NULL)
    return;

  for (sb = s->firstsub; sb != NULL; sb = sb->next)
    {
    if (! sb->output_done)
      {
      if (first)
        {
        out_startpobj();
        fprintf(destfile, "
|*************************************************************************
|* Autogenerated subsections for %s
|*************************************************************************
\n",             section);
        first = FALSE;
        }

      fprintf(destfile, "%s:\n", sb->name);
      out_pcall_ref(sb, FALSE);
      fprintf(destfile, "\n");
      }
    }
  }

/* Write out all remaining inheritance calls */
void out_all()
  {
  struct sectionst *s, *lastsec;
  int     first = TRUE, sout;

  if (! bp_found && find_section("before.program", TRUE) == NULL)
    {
    out_startpobj();
    fprintf(destfile, "
|*************************************************************************
|* Autogenerated sections/subsections for all remaining exported functions
|* from parent object.
|*
|* Note: before.program was not defined in this source and also is not
|*       in the parent, however, it must be added because this is where
|*       the parent dll object is loaded.
|*************************************************************************
\n");
    fprintf(destfile, "before.program:\n");
    out_dllload();
    fprintf(destfile, "\n");
    first = FALSE;
    }

  for (lastsec = NULL, s = enum_sections(); s != NULL; s = enum_next())
```

```
    {
    if (s->output_done) continue;

    if (first)
      {
      out_startpobj();
      fprintf(destfile, "
|*****************************************************************************
|* Autogenerated sections/subsections for all remaining exported functions
|* from parent object.
|*****************************************************************************
\n");
      first = FALSE;
      }

    /* Output the section id line, if this is the first time */
    if (s->secp != lastsec)
      {
      fprintf(destfile, "%s:\n", s->secp->name);
      lastsec = s->secp;
      }

    if (s->stype == S_SECTION)
      {
      if (strcmp(s->name, "before.program")  == 0)
        out_dllload();
      }
    else
      {
      /* Subsection name */
      fprintf(destfile, "%s:\n", s->name);
      }
    out_pcall_ref(s, FALSE);
    fprintf(destfile, "\n");
    }
  }

/* Add the _engen_object_version function to the script.  This must      */
/* be the last item output to the new source.  The function section      */
/* must be the last one so we're either in it or can add it because none */
/* previously existed.                                                   */
void out_ident(void)
  {
  char *t;
  int  major, minor;
  char dummyc;

  out_startpobj();

  if (strcmp(current_section, "functions") != 0)
    {
    fprintf(destfile, "functions:\n");
    strcpy(current_section, "functions");
    }

  /* Figure out a numeric number for the current version */
  /* This assume the version string will be like: "#.#" */
```

```
  sscanf(VERSION, "%d%c%d", &major, &dummyc, &minor);

  fprintf(destfile, "function extern long _engen_object_version()\n");
  fprintf(destfile, "{\n");
  fprintf(destfile, "\treturn(%d%02d)\n", major, minor);
  fprintf(destfile, "}\n");

  out_endpobj();
  }

/* Only used if optimizing function calls.  This is the function called  */
/* in before.program which will set all the parent dll function ids.     */
void out_getfids(void)
  {
  struct sectionst *s;

  if (! optimize_function_calls) return;

  out_startpobj();

  if (strcmp(current_section, "functions") != 0)
    {
    fprintf(destfile, "functions:\n");
    strcpy(current_section, "functions");
    }

  fprintf(destfile, "function _pobj_get_fids()\n");
  fprintf(destfile, "{\n");

  for (s = enum_sections(); s != NULL; s = enum_next())
    {
    fprintf(destfile, "\t_pobj_func_id = _pobj_get(");
    if (strlen(s->extfunc) <= 26)
      fprintf(destfile, "\"%s\")\n", s->extfunc);
    else
      fprintf(destfile, "\n\t                \"%s\")\n", s->extfunc);
    fprintf(destfile, "\t%s = _pobj_func_id\n", get_defname(s->extfunc));
    }
  fprintf(destfile, "}\n");
  }

void out_line(int skip_input)
  {
  struct commentst *c, *c2;

  out_endpobj();

  if (cmt_list != NULL)
    {
    for (c = cmt_list; c != NULL;)
      {
      fprintf(destfile, "%s\n", c->comment);

      c2 = c;
      c = c->next;
      free(c2->comment);
      free(c2);
```

```
      }
    cmt_list = NULL;
    cmt_last = NULL;
    }

  if (!skip_input)
    fprintf(destfile, "%s\n", src_line);
  }

/*****************************************************************************
 * Source input and main processing                                          *
 *****************************************************************************/

/* Determine if the string looks like a section/subsection line */
int looks_like_section(char *s)
  {
  for (; *s != EOS && isspace(*s); s++);  /* Skip leading whitespace */
  if (*s == EOS)
    return (FALSE);
  for (s++; *s != EOS && (isalnum(*s) || *s == '.'); s++);
  if (*s != ':')
    return (FALSE);
  for (s++; *s != EOS && isspace(*s); s++);  /* Verify remaining whitespace */
  if (*s != EOS)
    return (FALSE);

  return (TRUE);
  }

int is_section_name(char *s)
  {
  /* Determine if the passed name is the name of a Baan 4GL section name */

  /* Fixed session names */
  if (strcmp(s, "declaration") == 0 ||
      strcmp(s, "before.program") == 0 ||
      strcmp(s, "on.error") == 0 ||
      strcmp(s, "after.program") == 0 ||
      strcmp(s, "after.update.db.commit") == 0 ||
      strcmp(s, "functions") == 0 ||
      strcmp(s, "form.all") == 0 ||
      strcmp(s, "form.other") == 0 ||
      strcmp(s, "field.all") == 0 ||
      strcmp(s, "field.other") == 0 ||
      strcmp(s, "zoom.from.all") == 0 ||
      strcmp(s, "zoom.from.other") == 0 ||
      strcmp(s, "main.table.io") == 0)
     return (TRUE);

  /* Form session names */
  if (strpartcmp(s, "form.") == 0)
    {
    /* May only be followed by numeric values */
    for (s = strstr(s, ".") + 1; *s != EOS && isdigit(*s); s++);
    return (*s == EOS);
    }
```

```
  /* Choice, Field, and zoom from session names */
  if (strpartcmp(s, "choice.") == 0 ||
      strpartcmp(s, "field.") == 0 ||
      strpartcmp(s, "zoom.from.") == 0)
   {
   /* May only be followed by alpha numerics and a period */
   for (s = strstr(s, ".") + 1; *s != EOS && (isalnum(*s) || *s == '.'); s++);
   return (*s == EOS);
   }

  return (FALSE);
  }

int process_source(void)
  {
  char  destfullpath[PATHSIZE];
  char  objfullpath[PATHSIZE];
  char  src_copy[MAXLINE];
  char  *t, *p;
  char  t1[MAXLINE], t2[MAXLINE];
  int   cleaned;
  int   in_pobj;
  int   error;
  char  save_section[50];
  int   savelvl     = 0;     /* saved path search level of source */

  /* Split source name into component parts */
  strcpy(src_package, substr(src_name, 2, 3));
  strcpy(src_module, substr(src_name, 4, 6));
  strcpy(src_number, substr(src_name, 7, 10));
  sprintf(current_obj, "o%s%s%s", src_package, src_module, src_number);

  /* Locate the actual source file */
  if ((p = search_path(src_name, FALSE, FALSE)) == NULL)
    fatal_msg("source file %s cannot be found in your pacc", src_name);
  strcpy(srcfullpath, p);
  savelvl = lastlvl; /* Save the path search level for our source so */
                     /* we can track down the parent object later.   */


  /* Open source and output destination files */
  if ((srcfile = fopen(srcfullpath, "r")) == NULL)
    fatal_msg("unable to open %s for input", srcfullpath);
  sprintf(destfullpath, "%s.%s", srcfullpath, locasestr(MYNAME));
  if ((destfile = fopen(destfullpath, "w")) == NULL)
    fatal_msg("unable to open %s for output", destfullpath);

  in_pobj = FALSE;
  strcpy(current_section, "");
  strcpy(current_subsection, "");
  genobj = 0;        /* Initially off; use -1 to disable completely */
  cleaned = FALSE;   /* Whether or not we've cleaned old |#POBJ/|#END entries */
  error = FALSE;
  inhibit = FALSE;
  bp_found = FALSE;
  lineno = 0;
```

```
while (fgets(src_line, MAXLINE, srcfile) != NULL)
  {
  chop(src_line);

  /* Strip out all |#pobj/|#end directives atuomatically; we'll readd */
  /* if necessary anyway.                                              */
  strcpy(src_copy, src_line);
  t = strtok(src_copy, WHITESPACE);
  if (t != NULL)
    {
    strcpy(t1, locasestr(t));
    }
  else
    strcpy(t1, "");
  if (strcmp(t1, "|#pobj") == 0)
    {
    in_pobj = TRUE;
    cleaned = TRUE;
    continue;
    }
  if (in_pobj)
    {
    if (strcmp(t1, "|#end") == 0)
      in_pobj = FALSE;
    continue;
    }

  lineno ++;

  /* Found a section/subsection separator */
  if (strcmp(t1, "default:") != 0 && looks_like_section(src_line))
    {
    chop(t1);  /* Strip trailing ":" as it's no longer needed */

    /* If previous section name was functions then we've got an error as */
    /* we've encountered a new section (or sub-section) name.            */
    if (strcmp(current_section, "functions") == 0)
      {
      warning_msg("potential section/subsection found after "
              "'functions'; ignored");
      out_line(FALSE);
      continue;
      }

    /* If we've already scanned a section/subsection and are generating  */
    /* object inheritance code, do so now before going into the next     */
    /* section.                                                          */
    if (current_section[0] != EOS && genobj > 0)
      {
      if (! out_pcall(current_section, current_subsection, FALSE))
        error = 1;
      }

    /* Always reset the inhibit flag on each section/subsection change   */
    /* Must do it here because we don't want to inhibit auto-generated   */
    /* code for undeclared sections/subsections.                         */
    inhibit = FALSE;
```

```
      strcpy(save_section, current_section);

      if (is_section_name(t1))
        {
        /* On section name change, we must output all the remaining        */
        /* subsection calls for interfacing to the parent object.          */
        if (save_section[0] != EOS && genobj > 0)
          out_rest(save_section);

        strcpy(current_section, t1);
        strcpy(current_subsection, ""); /* No subsection or not known yet  */
        }
      else
        strcpy(current_subsection, t1);

      /* Declaration section must be the first one! */
      if (strcmp(current_section, "declaration") == 0 && save_section[0] !=
EOS)
        {
        warning_msg("declaration section must be first section "
                "(no generation)");
        genobj = -1;
        }

      /* If we're entering the before.program section, then              */
      /* output the section name (and comments) and the dll load code    */
      if (strcmp(current_section, "before.program") == 0 && genobj > 0)
        {
        out_line(FALSE);
        out_dllload();
        continue;
        }

      /* If we're entering the function section then output everything   */
      /* from the exported function list that hasn't been done yet.      */
      if (strcmp(current_section, "functions") == 0 && genobj > 0)
        out_all();
      }

    if (strcmp(t1, "|#inhibit") == 0)
      {
      out_line(FALSE);
      inhibit = TRUE;
      continue;
      }

    if (strcmp(t1, "|#call") == 0 && genobj > 0)
      {
      /* The |#call directive was found, if in a valid section/subsection */
      /* attempt to generate the code to execute the parent object --     */
      /* if the parent doesn't have any function for this then we'll just */
      /* ignore this -- it's a comment then.                              */
      out_line(FALSE);
      if (current_section[0] == EOS ||
          strcmp(current_section, "delcaration") == 0)
        {
```

```
      error_msg("|#CALL must be in a non-declaration section");
      error = 1;
      break;
      }
    if (! out_pcall(current_section, current_subsection, TRUE))
      error = 1;

    continue;
    }

  /* Found the |#parent directive */
  if (strcmp(t1, "|#parent") == 0 || strpartcmp(t1, "|#parent ") == 0)
    {
    /* Can only be defined once and must be defined in the declaration  */
    /* section.                                                         */
    if (genobj == 1)
      {
      error_msg("|#PARENT directive already declared");
      error = 1;
      break;
      }

    if (strcmp(current_section, "declaration") != 0)
      {
      error_msg("|#PARENT directive must be in declaration section");
      error = 1;
      break;
      }

    /* Process the |#PARENT directive */
    genobj = 1;    /* Flag that we're now doing object generation */
    t = nexttok(WHITESPACE);
    if (t != NULL)
      {
      strcpy(parent_obj, t);

      /* Verify user used correct naming convention "oppmmmnnnn" */
      if (strlen(parent_obj) != 10 ||
          parent_obj[0] != 'o'     ||
          ! islower(parent_obj[1]) ||
          ! islower(parent_obj[2]) ||
          ! islower(parent_obj[3]) ||
          ! islower(parent_obj[4]) ||
          ! islower(parent_obj[5]) ||
          ! isdigit(parent_obj[6]) ||
          ! isdigit(parent_obj[7]) ||
          ! isdigit(parent_obj[8]) ||
          ! isdigit(parent_obj[9]))
        {
        error_msg("|#PARENT follwed by invalid object name");
        error = 1;
        break;
        }

      /* Parent must be in same package */
      if (strncmp(current_obj, parent_obj, 3) != 0)
        {
```

```
        error_msg("|#PARENT refers to object in a different package");
        error = 1;
        break;
        }
      }
    else
      {
      sprintf(parent_obj, "o%s%s%s", src_package, src_module, src_number);
      }

    /* Find the parent object; remember to start above the source's      */
    /* current locatin in the VRC pathlist.                              */
    if ((p = search_path(parent_obj, savelvl, TRUE)) == NULL)
      fatal_msg("parent object %s cannot be found in your pacc", parent_obj);
    strcpy(objfullpath, p);
    printf("%s: Inheriting from: %s\n", MYNAME, objfullpath);

    /* Load all the exported section/subsections from the parent object */
    if (! loadparent(objfullpath))
      {
      error = 1;
      break;
      }

    /* Output the required variables declarations for loading the        */
    /* parent dll and making the function calls.                         */
    out_line(FALSE);
    out_startpobj();
    fprintf(destfile, "\tlong\t_pobj_dll_id\n");
    fprintf(destfile, "\tlong\t_pobj_func_id\n");
    fprintf(destfile, "\tlong\t_pobj_err\n");
    fprintf(destfile, "\tstring\t_pobj_path(255)\n");
    out_functiondefs();
    fprintf(destfile, "\t#define _pobj_get(a)\t"
                      "get_function(_pobj_dll_id,a)\n");
    fprintf(destfile, "\t#define _pobj_exe0(a)\t_pobj_err = "
                      "exec_function(_pobj_dll_id,a)\n");
    fprintf(destfile, "\t#define _pobj_exe1(a)\t{ "
                      "_pobj_func_id = get_function(_pobj_dll_id,a)\n");
    fprintf(destfile, "^\t\t\t\t_pobj_err = exec_function(_pobj_dll_id,\n");
    fprintf(destfile, "^\t\t\t\t\t    _pobj_func_id) }\n");
    continue;
    }

  /* Save all comments and blanks; these should be kept with the */
  /* following line. */
  if (*t1 == EOS || *t1 == '|')
    {
    add_comment(src_line);
    continue;
    }

  /* Print the line from the source file as is */
  out_line(FALSE);
  }

/* Output the final section call code */
```

```
    if (current_section[0] != EOS && genobj > 0)
      {
      out_pcall(current_section, current_subsection, FALSE);
      inhibit = FALSE;
      out_rest(current_section);
      }
    /* Output all the rest of the sections exported from the parent object */
    if (genobj > 0)
      {
      inhibit = FALSE;
      out_all();

      out_getfids();
      out_ident();
      }

    /* Output any remaining comments/blank lines */
    out_line(TRUE);

    fclose(srcfile);
    fclose(destfile);

    /* If an error occurred, unlink the temporary object generated source and */
    /* skip calling Baan's std_gen routine (which will also cause the compile */
    /* step to abort as there's no generated source available for it.          */
    if (error)
      unlink(destfullpath);
    else
      {
      /* Ok! Change the source file to be our newly generated file. */
      if (genobj > 0 || cleaned)
        {
        unlink(srcfullpath);
        rename_file(destfullpath, srcfullpath);
        }
      else
        {
        unlink(destfullpath);  /* No changes, scrap this file */
        }
      }

    return (!error);
    }

/*****************************************************************************
 * main function                                                             *
 *****************************************************************************/

void get_base_env(void)
  {
  char *s;
  struct passwd *pw;

  /* Get environment strings */
  if ((s = getenv("BSE")) == NULL)
    fatal_msg("could not determine the BSE");
  strcpy(bse, s);
```

```
  if ((s = getenv("BSE_TMP")) == NULL)
    {
    strcpy(bsetmp, bse);
    strcat(bsetmp, "/tmp");
    }
  else
    strcpy(bsetmp, s);
  if ((s = getenv("USER")) == NULL)
    {
#ifdef WINNT
#else
    if ((pw = getpwuid(getuid())) == NULL)
      fatal_msg("unable to retrieve user information");
    strcpy(user, pw->pw_name);
#endif
    }
  else
    strcpy(user, s);
  }

void get_paths(void)
  {
  #define SPECIALSIZE 4096
  char *s;
  FILE *fd;
  char rec[SPECIALSIZE];
  char pathlist[SPECIALSIZE];
  char pathcode[10];

  if (pacc[0] == EOS)
    {
    if ((s = getenv("PACKAGE_COMB")) == NULL)
      {
      /* Retrieve package combination from user file */
      sprintf(command, "%s/lib/user/u%s", bse, user);
      if ((fd = fopen(command, "r")) == NULL)
      fatal_msg("cannot open user file for %s", user);
      while (fgets(rec, SPECIALSIZE, fd) != NULL)
      {
      chop(rec);
      if (strpartcmp(rec, "pacc:") == 0)
        {
        strtok(rec, ":");
        strcpy(pacc, nexttok(WHITESPACE));
        break;
        }
      }
      fclose(fd);
      }
    else
      strcpy(pacc, s);
    }
  if (pacc[0] == EOS)
    fatal_msg("could not determine what package combination to use");

  /* Retieve all path lists from the fd6.1.xxxxx file */
  sprintf(command, "%s/lib/fd%s.%s", bse, BAANTOOLSVER, pacc);
```

```
    if ((fd = fopen(command, "r")) == NULL)
      fatal_msg("cannot open pacc fd file for %s", pacc);
    while (fgets(rec, SPECIALSIZE, fd) != NULL)
      {
      chop(rec);
      if ((s = strstr(rec, ":")) != NULL)
        {
        *s = EOS;
        s++;
        add_path(rec, s);
        }
      }
    fclose(fd);
    }

  void end_program(int exitcode)
    {
    FILE *baanout;
    FILE *engenout;
    FILE *tempfile;
    char rec[MAXLINE];
    char baan4c_work[PATHSIZE];

    if (baan4c_output[0] != EOS)
      {
      /* Reset stdout and stderr to the original output */
      fclose(stdout);
      dup(save_stdout);
      fclose(stderr);
      dup(save_stderr);
      close(save_stderr);
      close(save_stdout);
      tempfile = fdopen(STDOUT, "w");
      memcpy(stdout, tempfile, sizeof(FILE));
      tempfile = fdopen(STDERR, "w");
      memcpy(stderr, tempfile, sizeof(FILE));

      sprintf(baan4c_work, "%s.%d", baan4c_output, getpid());
      sprintf(command, "mv %s %s", baan4c_output, baan4c_work);
      system(command);
      sprintf(command, "mv %s %s", temp_output, baan4c_output);
      system(command);

      /* Transfer the contents of the std_gen output into the work file
         that was created by our program so Baan will pick it all up as
         one output and show this to the developer */

      baanout = fopen(baan4c_work, "r");
      engenout = fopen(baan4c_output, "a");
      while (fgets(rec, MAXLINE, baanout) != NULL)
        {
        fputs(rec, engenout);
        }
      fclose(baanout);
      fclose(engenout);
      unlink(baan4c_work);
      }
```

```
  dispose_sections();
  dispose_paths();
  dispose_comments();

  exit (exitcode);
  }

/* Set stdout to unbuffered; do this by closing it and reopening to the
   same place */
void setnobuf(FILE *buffile, char *mode)
  {
  int hold_fd;
  int orig_fd;
  FILE *newfile;

  orig_fd = fileno(buffile);
  hold_fd = dup(orig_fd);
  fclose(buffile);
  dup2(hold_fd, orig_fd);
  close(hold_fd);
  newfile = fdopen(orig_fd, mode);
  setbuf(newfile, NULL);

  memcpy(buffile, newfile, sizeof(FILE));
  }

int main(int argc, char **argv)
  {
  int i;

  setnobuf(stdout, "w");

  info_msg("Version %s", VERSION);
  info_msg(COPYRIGHT);

#ifdef WINNT
  strcpy(bic_info, "bic_info.exe");
  strcpy(std_gen, "std_genr.exe");
#else
  sprintf(bic_info, "bic_info%s", BAANTOOLSVER);
  sprintf(std_gen, "std_gen%s.real", BAANTOOLSVER);
#endif

  get_base_env();

  /* Scan the command line; we're using std_gen6.1 command line so look */
  /* for "-s source".  We're interested in the source file name and the */
  /* output direction parameters (-qe) that is now used under Baan IVc. */
  strcpy(src_name, "");
  strcpy(baan4c_output, "");
  strcpy(pacc, "");
  for (i = 1; i < argc; i++)
    {
    if (strcmp(argv[i], "-s") == 0)
      {
      i++;
```

```
        if (i < argc)
          strcpy(src_name, argv[i]);
        }
      else if (strcmp(argv[i], "-pacc") == 0)
        {
        /* Use this for the package combination name */
        i++;
        if (i < argc)
        {
        strcpy(pacc, argv[i]);
        }
        }
      else if (strcmp(argv[i], "-qe") == 0)
        {
        i++;
        if (i < argc)
          {
          int dupout;

          strcpy(baan4c_output, argv[i]);
          sprintf(temp_output, "%s/tmp%s.%d", bsetmp, MYNAME, getpid());

          /* Redirect stdout and stderr to a file so it can be placed in the */
          /* output file for Baan to display to the developer properly.      */
          save_stdout = dup(STDOUT);
          save_stderr = dup(STDERR);
          if (freopen(temp_output, "w", stdout) == NULL)
            fatal_msg("unable to reopen stdout");
          dup2(STDOUT, STDERR);

        info_msg("Version %s", VERSION);
        info_msg(COPYRIGHT);
          }
        }
      }
    if (src_name[0] == EOS)
      fatal_msg("source file name not found on command line");

    get_paths();
    if (process_source())
      {
      int status;
      pid_t child;
      char  orig_name[FILENAMESIZE];

      /* Call Baan's std_gen program to process the source file */
      if (genobj > 0)
        info_msg("Executing std_gen on %s source", MYNAME);
      else
        info_msg("Executing std_gen on normal source");

      if ((child = fork()) == 0)
        {
        sprintf(command, "%s/bin/%s", bse, std_gen);
#ifdef WINNT
        strcpy(orig_name, "stdgen.exe");
#else
```

```
        sprintf(orig_name, "std_gen%s", BAANTOOLSVER);
#endif
        argv[0] = orig_name;
        execvp(command, argv);
        fatal_msg("unable to execute %s", std_gen);
        exit(1);
        }
      waitpid(child, &status, 0);
      end_program(status/256);
      }

  end_program(1);
  }
```

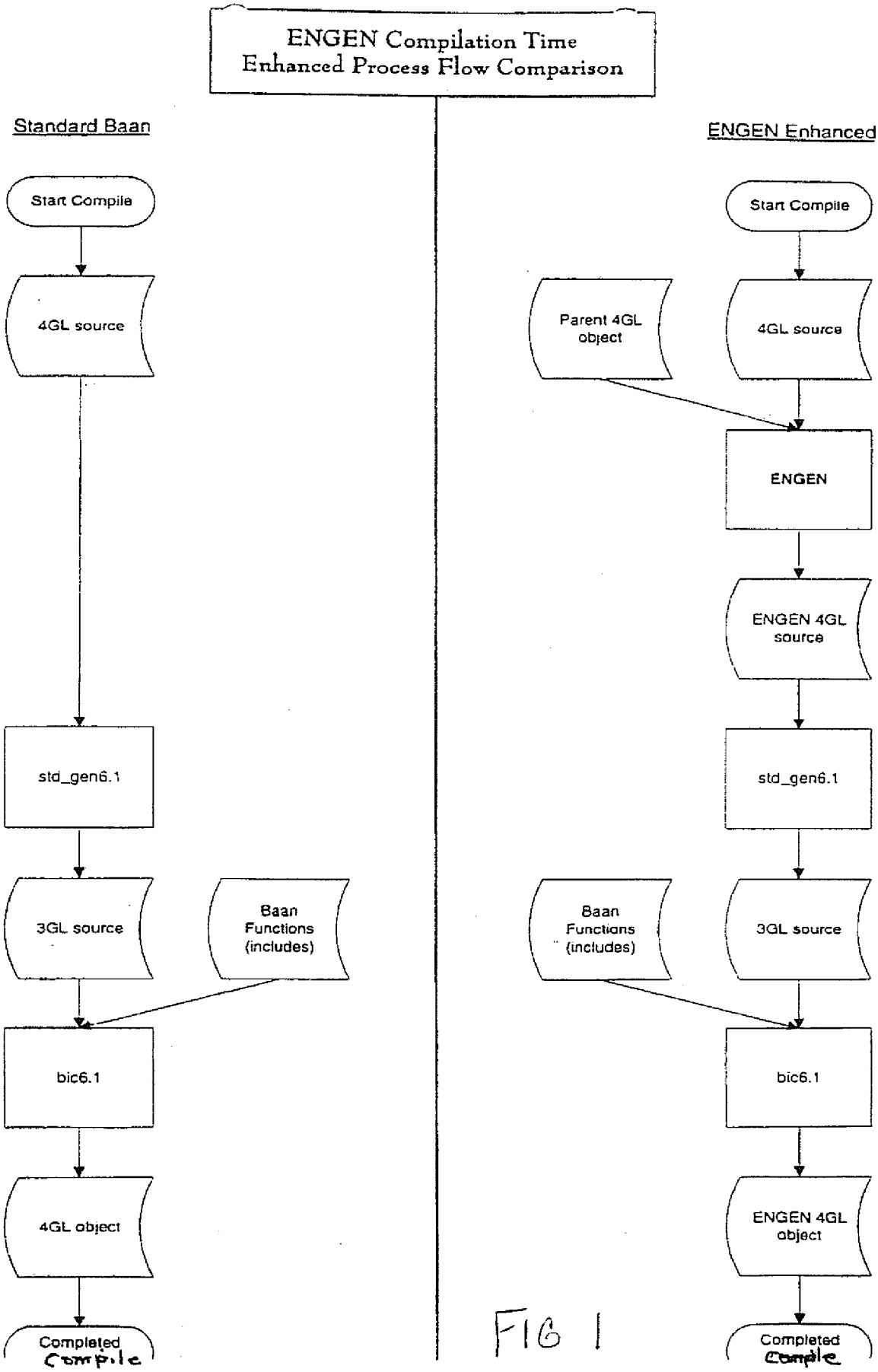


FIG 1

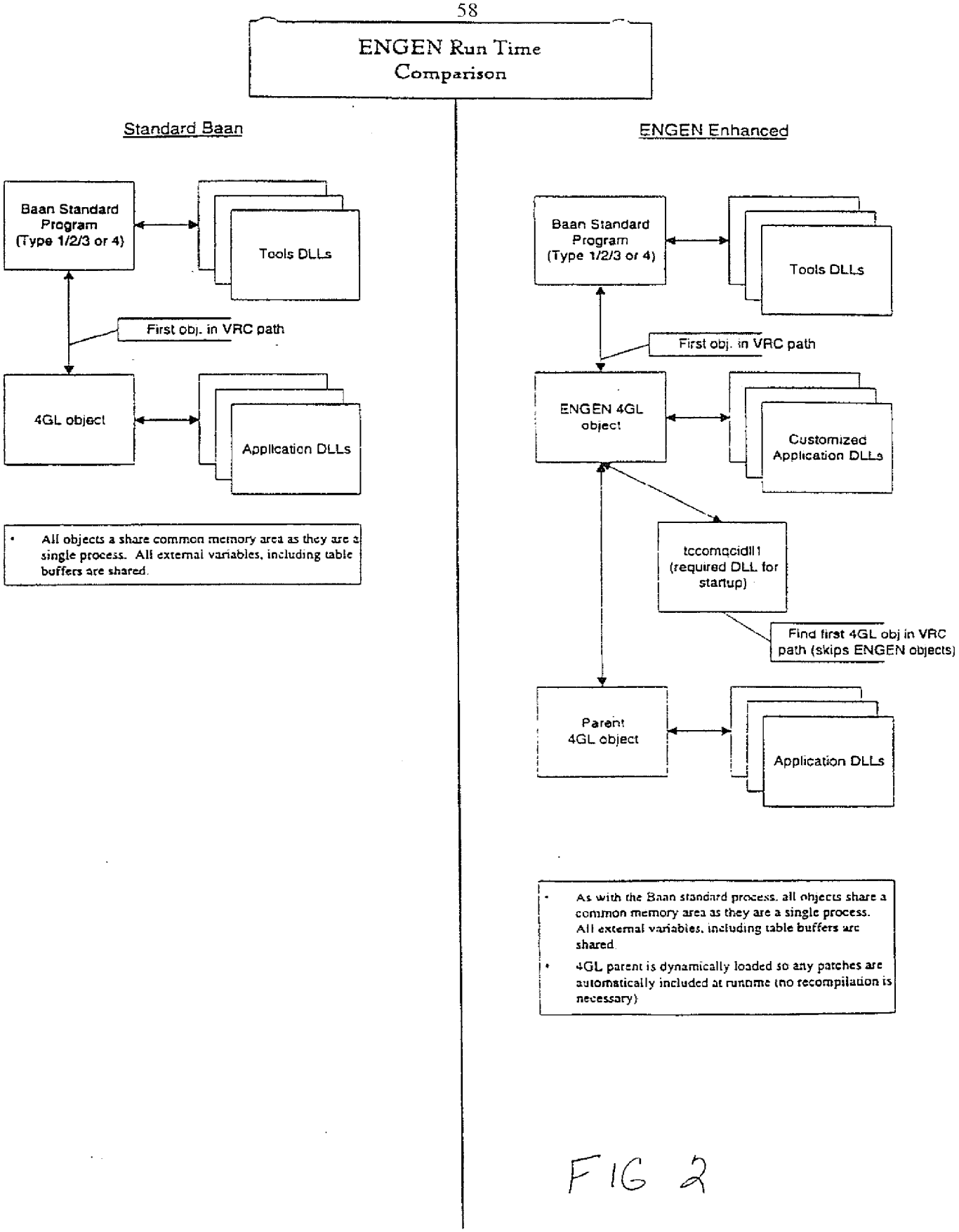

ENGEN Run Time Comparison
Standard Baan
Baan Standard Program (Type 1/2/3 or 4)
Tools DLLs
First obj. in VRC path
4GL object
Application DLLs
All objects a share common memory area as they are a single process. All external variables, including table buffers are shared.
ENGEN Enhanced
Baan Standard Program (Type 1/2/3 or 4)
Tools DLLs
First obj. in VRC path
ENGEN 4GL object
Customized Application DLLs
tccomqcidll1 (required DLL for startup)
Find first 4GL obj in VRC path (skips ENGEN objects)
Parent 4GL object
Application DLLs
As with the Baan standard process, all objects share a common memory area as they are a single process. All external variables, including table buffers are shared.
4GL parent is dynamically loaded so any patches are automatically included at runtime (no recompilation is necessary)

FIG 2

ENGEN Logic Overview

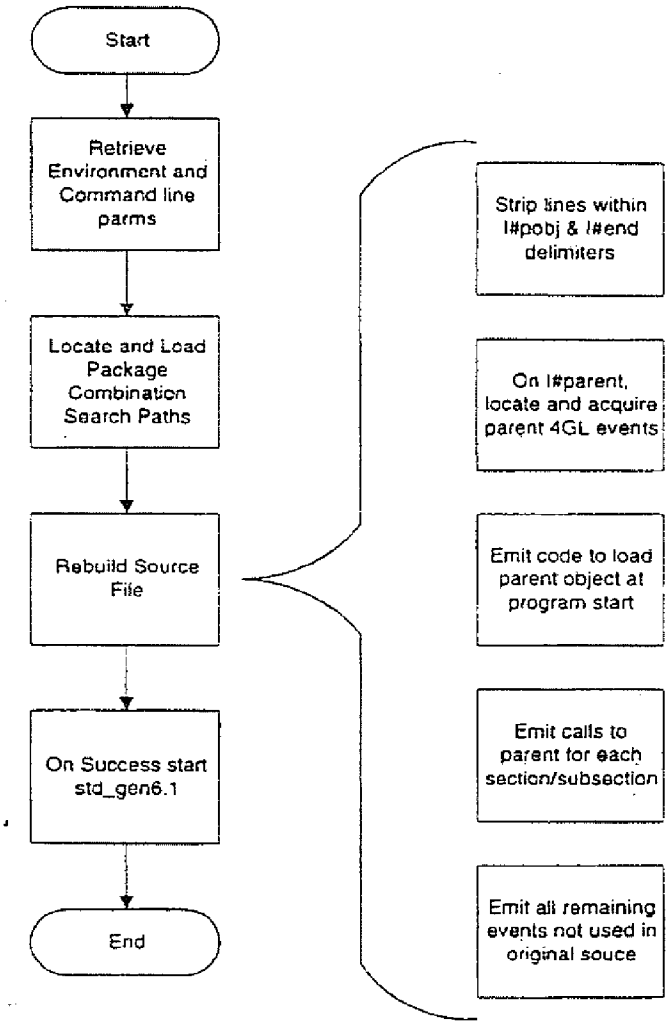
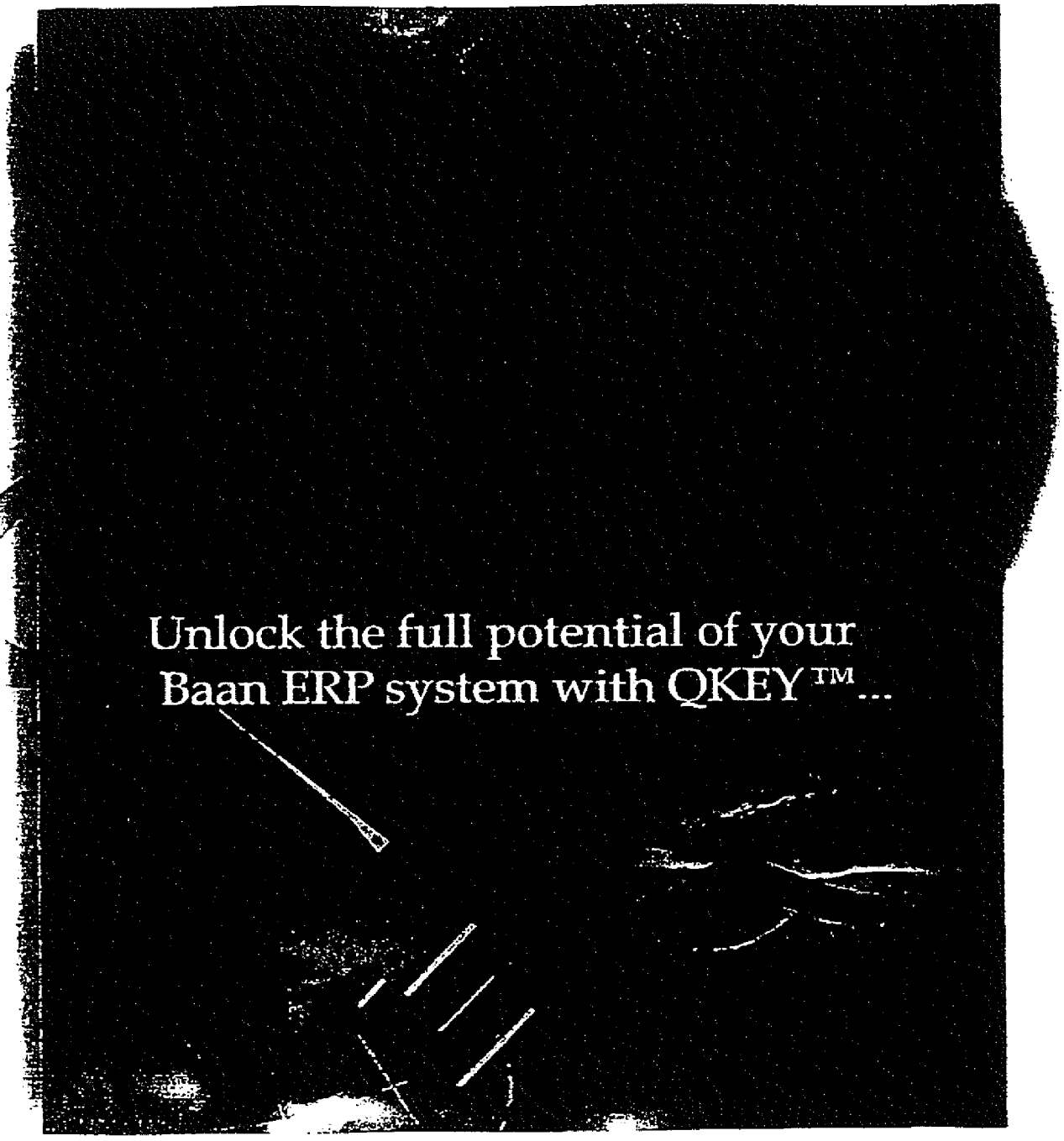
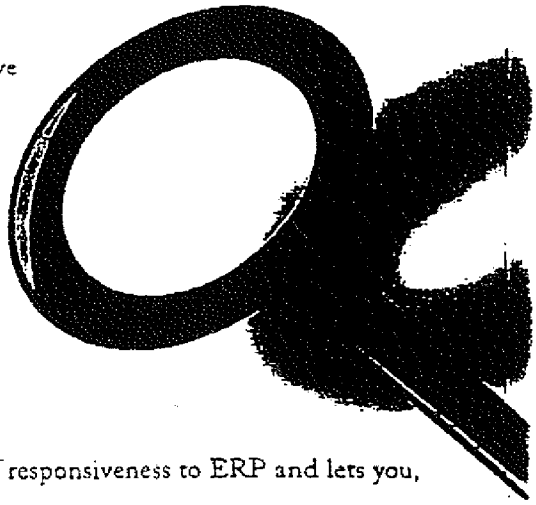


FIG 3



BAAN

An Enterprise Resource Planning solution is no small investment. So it had better open some new doors. Above all, your ERP system should enable you to extend your capabilities, expand your horizons and respond to a dynamic environment. It must be robust and flexible, and leave you in control.

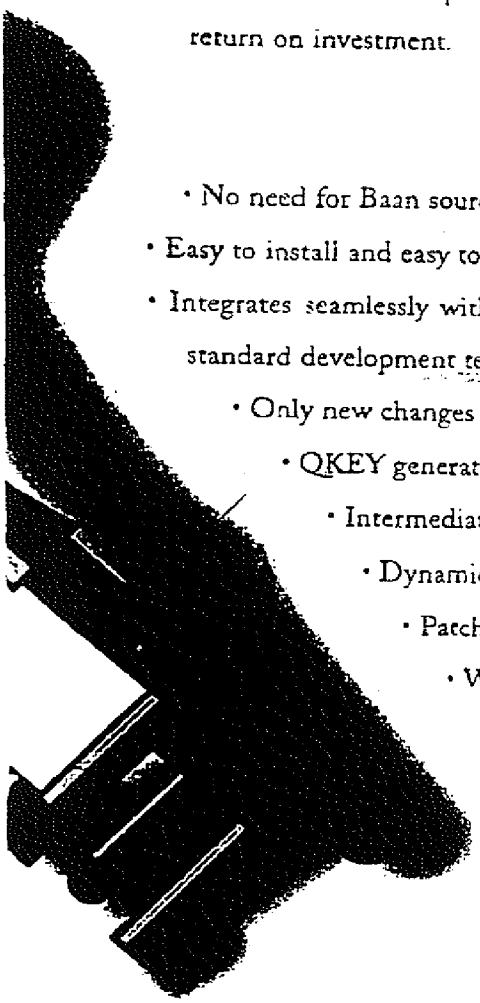


Quality Consultants Inc. (QCI) is committed to providing innovative tools and proven expertise that allow businesses to optimize the performance of their ERP systems. Drawing upon our in-depth experience in ERP implementation and systems integration, QCI has developed a revolutionary development tool – QKEY – that lets you modify and enhance your Baan ERP system *without using Baan source code*.

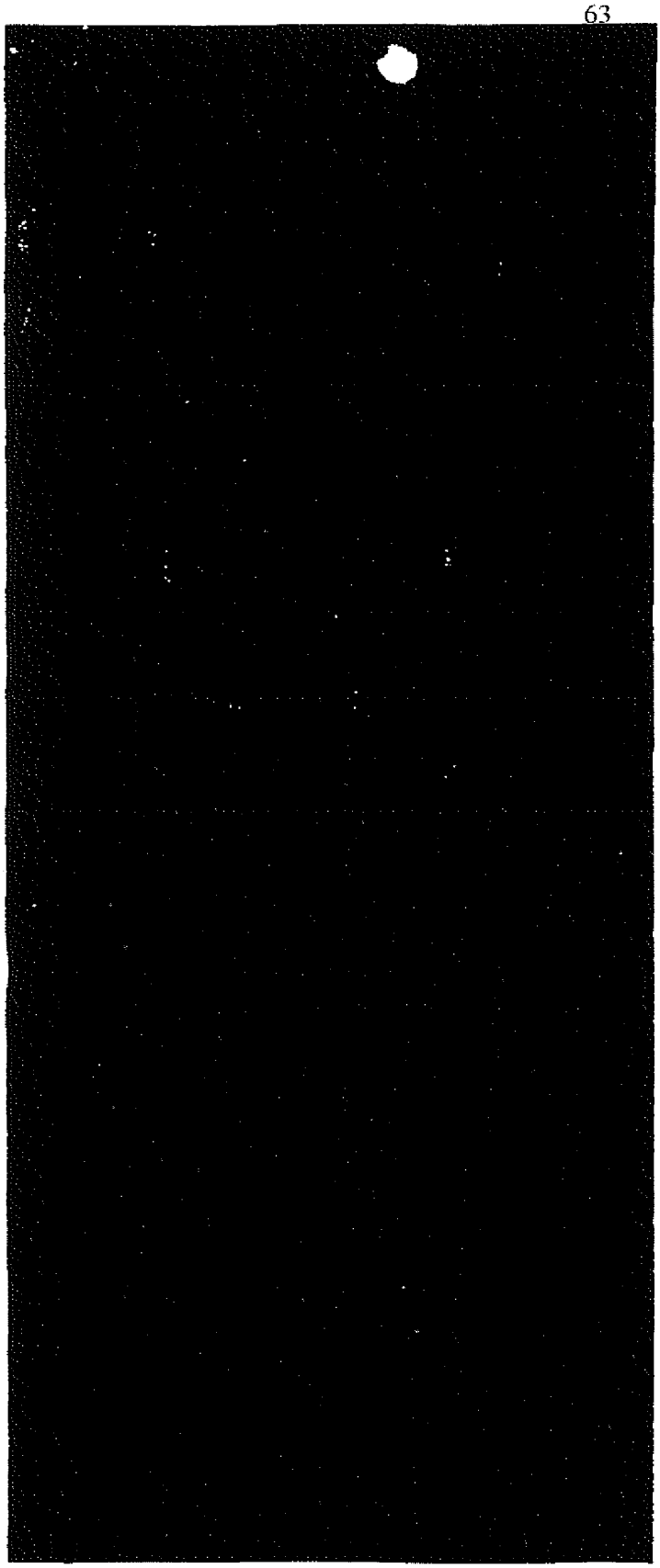
QKEY Object Applet Integrator for Baan adds a new dimension of responsiveness to ERP and lets you, not your software, set the agenda and the pace. Acting as a bridge between a manufacturer's specific needs and the capabilities of the Baan system, QKEY is a unique software utility that makes customization and upgrades easier, faster and more economical. You can enhance functionality and refine your system to meet the specific demands of your business. Reduce testing time during the upgrading process. Even better, all your developers need is Baan Tools knowledge, freeing you from the need to hire expensive source code specialists.

At the core of QKEY is a preprocessor for Baan source scripts. Developers can use QKEY to manipulate 4GL scripts to enhance current Baan sessions without having the source code available. QKEY makes this possible by allowing the user to view each Baan standard session as an object, and by permitting the overriding or extension of this object's associated events (or *public methods* in Object Oriented Programming terminology). The result? You can create customized 4GL objects that contain only your new code so that when a Baan patch or Version Release Control (VRC) is installed your changes can be reused automatically. In most cases, there is no need to recompile customizations. Even in installations that have already purchased Baan's source code, these features can save you considerable time and money.

Significantly lowered progr ming costs, ease of implementation and red d development time are just the beginning. With compressed integration cycles and an information technology infrastructure that's customized to your unique needs, you're free to meet the distinctive demands of your supply chain faster and more accurately. In a marketplace where speed and agility are nothing short of critical, QKEY's benefits make perfect sense. What's more, QKEY is priced low to guarantee you a rapid return on investment.



- No need for Baan source code when making enhancements.
- Easy to install and easy to use.
- Integrates seamlessly with the Baan development environment and does not interfere with standard development techniques.
- Only new changes are stored in customized scripts.
- QKEY generator automatically searches the VRC paths to find the standard object.
- Intermediate QKEY objects in the VRC tree are skipped.
- Dynamically searches for and loads the standard object when a session starts.
- Patched objects are automatically used if found first in the VRC path.
- Works with 4GL scripts.
- Baan debugger can still be used.

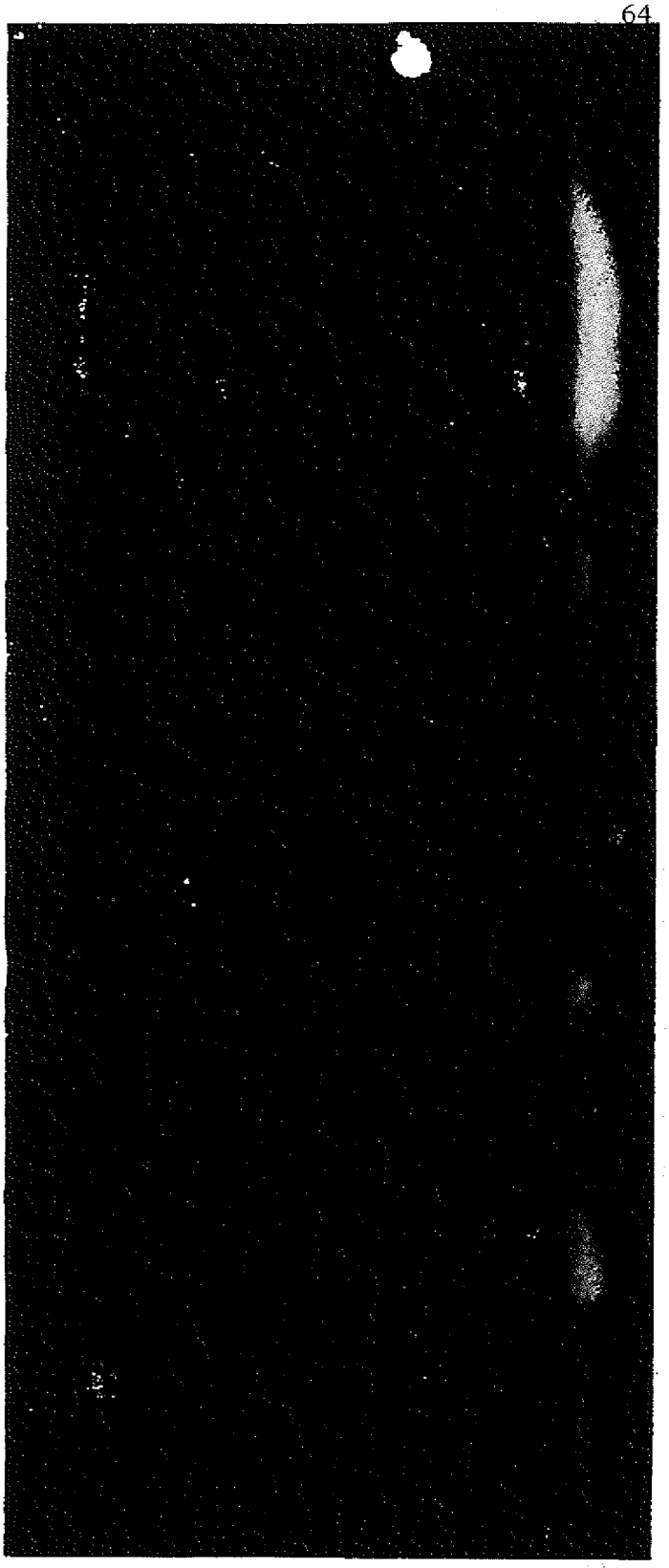


"A major customization for us was pricing. Our costs and selling prices are driven by commodity prices, which change daily. QKEY allowed us to design, develop and implement our unique pricing methodology without modifying Baan source code. This was very important to us because we wanted to use a system that allowed us to benefit from upgrades and still customize the system to fit our business."

"QKEY has enhanced our ability to migrate from the legacy system to Baan, and without it, we could not have delivered even a partial implementation in our time frame. I am very satisfied with the product."

Joy Conner
Programmer Analyst
JC Nordt

Quality Consultants Inc. (QCI) specializes in the implementation and support of Baan's Enterprise Resource Planning software solutions. QCI is a licensed Baan Service Partner and a member of the Process Technology Group of companies.



"QKEY is an excellent product that makes it easy to modify scripts without purchasing Baan source code. This saves our company money and allows us to make the modifications necessary to fit our business. With QKEY, we are able to make enhancements to Baan scripts, including fields and specific sections within the script. We are very satisfied with QKEY."

Phil Chesher

Systems Analyst

The Andersons

WHAT IS CLAIMED IS:

1. An apparatus for preprocessing existing source scripts, comprising:

a) a computer having a memory storage device and a microprocessor;

b) an operating system stored in said memory storage device;

c) means for viewing an object; and,

d) means for permitting the overriding of the associated events of said object.

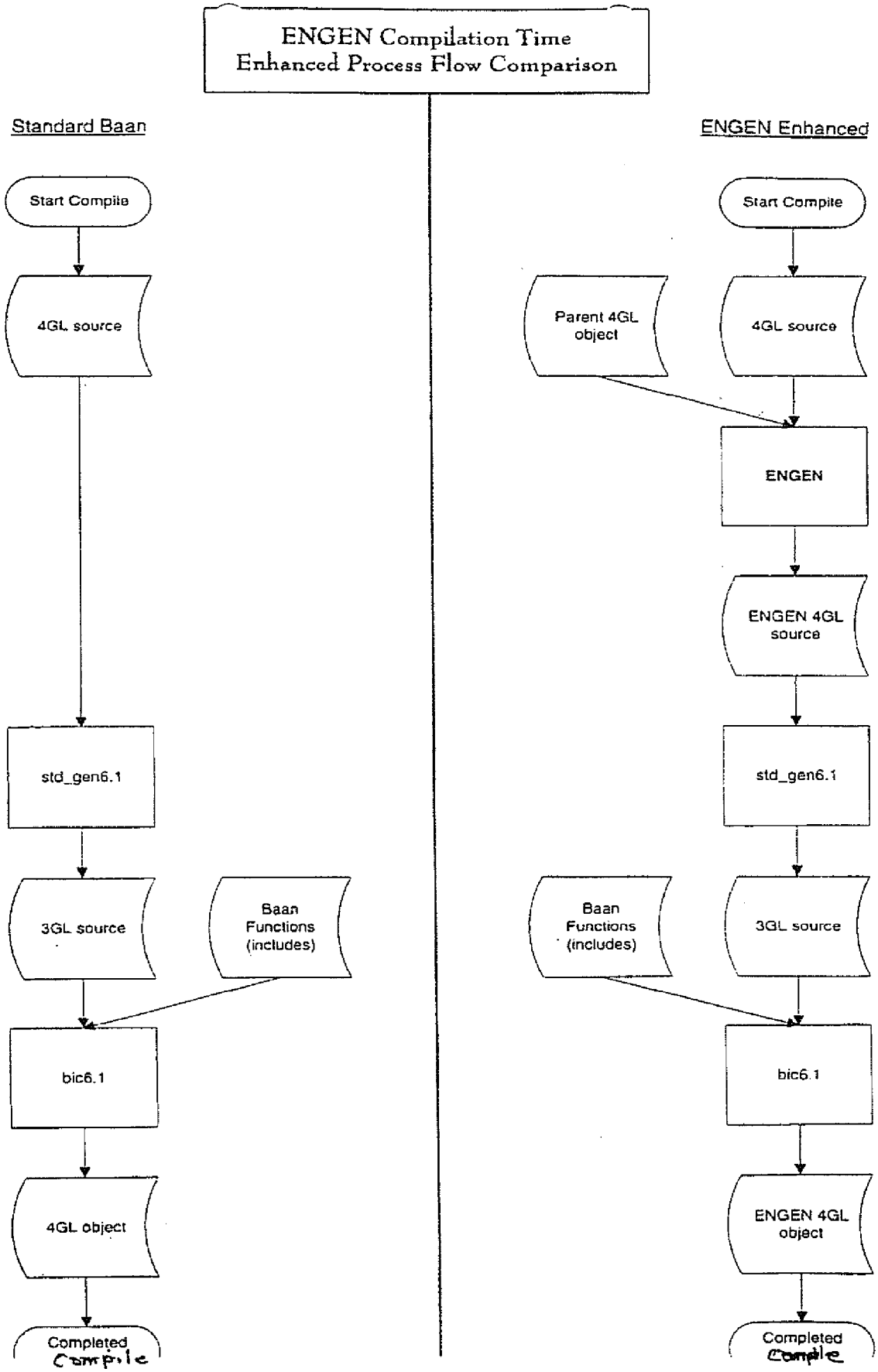


FIG. 1

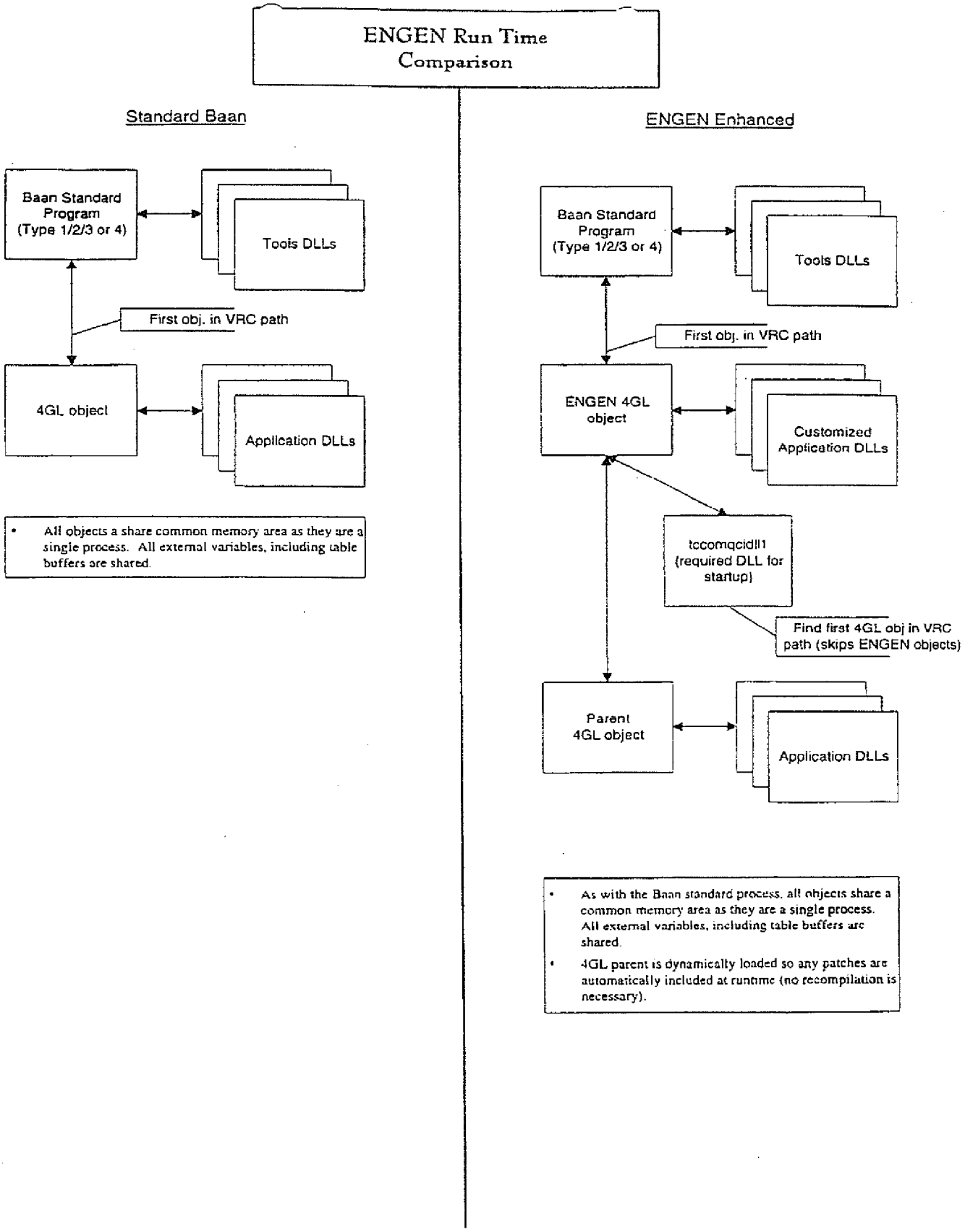

ENGEN Run Time Comparison
Standard Baan
Baan Standard Program (Type 1/2/3 or 4)
Tools DLLs
First obj. in VRC path
4GL object
Application DLLs
All objects a share common memory area as they are a single process. All external variables, including table buffers are shared.
ENGEN Enhanced
Baan Standard Program (Type 1/2/3 or 4)
Tools DLLs
First obj. in VRC path
ENGEN 4GL object
Customized Application DLLs
tccomqcidll1 (required DLL for startup)
Find first 4GL obj in VRC path (skips ENGEN objects)
Parent 4GL object
Application DLLs
As with the Baan standard process, all objects share a common memory area as they are a single process. All external variables, including table buffers are shared.
4GL parent is dynamically loaded so any patches are automatically included at runtime (no recompilation is necessary).

FIG 2

ENGEN Logic Overview

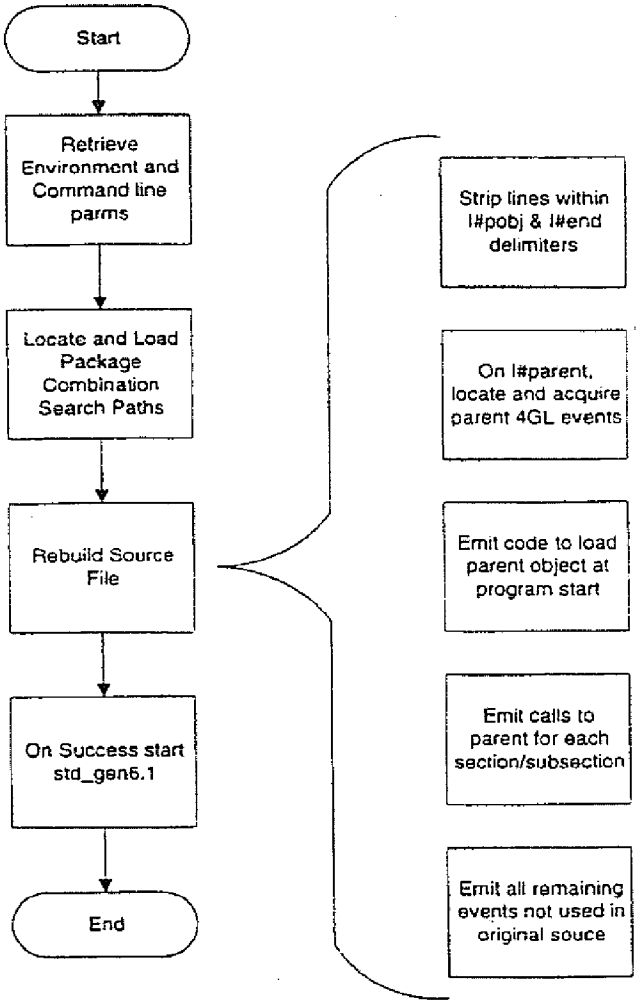


FIG 3

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/44 — A3

(11) International Publication Number: **WO 99/63431**

(43) International Publication Date: 9 December 1999 (09.12.99)

(21) International Application Number: PCT/US99/12075

(22) International Filing Date: 28 May 1999 (28.05.99)

(30) Priority Data:
60/087,440 1 June 1998 (01.06.98) US

(71) Applicant *(for all designated States except US)*: QUALITY CONSULTANTS, INC. [US/US]; Suite 215, 1775 The Exchange, Atlanta, GA 30339 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: BROCK, Kevin, R. [US/US]; 1277 Peninsula Trail, Lawrenceville, GA 30044 (US).

(74) Agent: BERNSTEIN, Jason, A.; Bernstein & Associates, P.C., Suite 121, 30 Perimeter Center East, Atlanta, GA 30346–1902 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

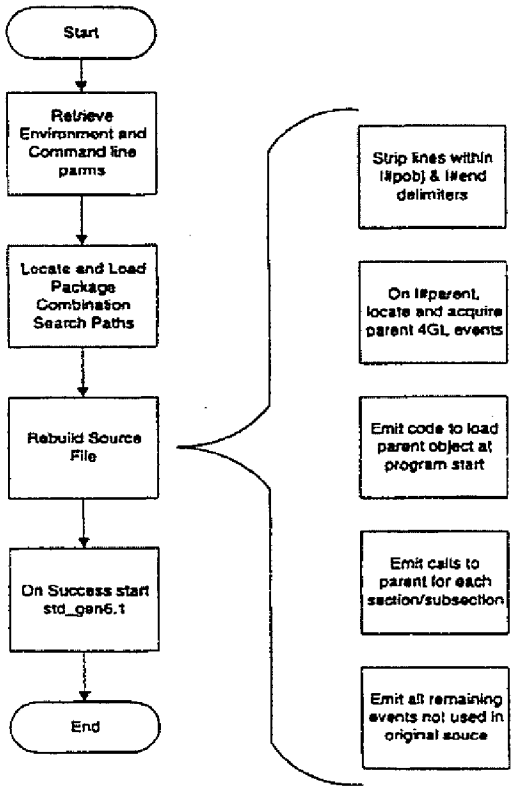
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(88) Date of publication of the international search report:
17 February 2000 (17.02.00)

(54) Title: PREPROCESSOR FOR ENCAPSULATING SOFTWARE SOURCE SCRIPTS

(57) Abstract

In BAAN software, a user defined script is created by copying a standard script to a custom directory and making the required changes to the standard script so as to obtain a custom script. The problem with this method is that each time the standard script is patched, the custom script will need to be changed manually with the same fixes. The invention, in the form of the Qkey preprocessor, provides a solution to this problem. First, the user creates a custom script containing only the user's add-ons and modifications to the standard script, whereby preprocessor directives are used to call or override code contained in the standard script. Second, the Qkey preprocessor uses the directives to automatically add to the custom script the BAAN software instructions necessary to make the appropriate calls to and to dynamically load the standard script. Finally, the user script is compiled.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 99/12075

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	QUALITY CONSULTANTS: "QKEY 2.10 User and Technical Guide" QKEY ENCAPSULATED GENERATOR FOR BAAN, 12 April 1998 (1998-04-12), page 1 XP002124554 the whole document	1
X	US 5 754 862 A (JONES DAVID T ET AL) 19 May 1998 (1998-05-19) column 2, line 62 -column 3, line 10 column 7, line 65 -column 9, line 10 column 10, line 23 -column 11, line 35	1

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 December 1999	22/12/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bijn, K

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

ernational Application No
PCT/US 99/12075

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5754862 A	19-05-1998	US 5410705 A	25-04-1995
		US 5297284 A	22-03-1994
		US 5854931 A	29-12-1995

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 9/44

A3

(11) International Publication Number: WO 99/63431

(43) International Publication Date: 9 December 1999 (09.12.99)

(21) International Application Number: PCT/US99/12075

(22) International Filing Date: 28 May 1999 (28.05.99)

(30) Priority Data:
60/087,440 1 June 1998 (01.06.98) US

(71) Applicant *(for all designated States except US)*: QUALITY CONSULTANTS, INC. [US/US]; Suite 215, 1775 The Exchange, Atlanta, GA 30339 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: BROCK, Kevin, R. [US/US]; 1277 Peninsula Trail, Lawrenceville, GA 30044 (US).

(74) Agent: BERNSTEIN, Jason, A.: Bernstein & Associates, P.C., Suite 121, 30 Perimeter Center East, Atlanta, GA 30346-1902 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

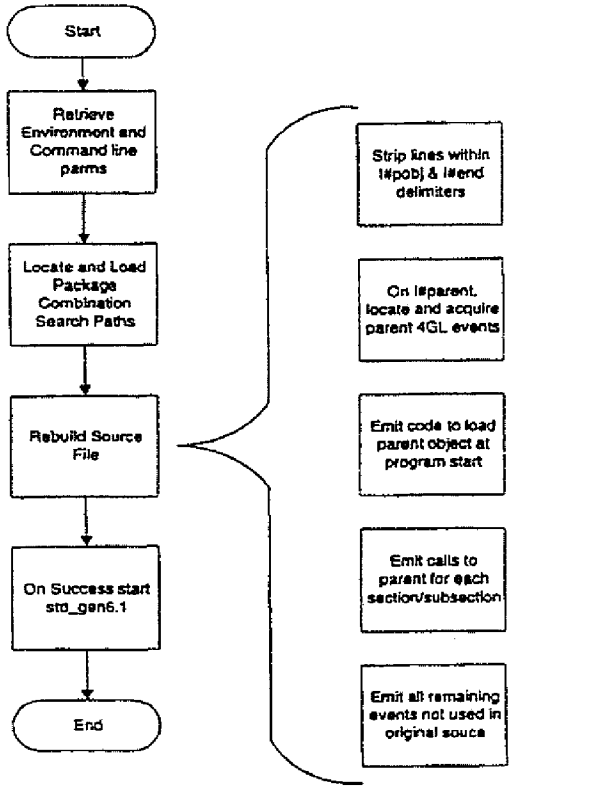
Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(88) Date of publication of the international search report:
17 February 2000 (17.02.00)

(54) Title: PREPROCESSOR FOR ENCAPSULATING SOFTWARE SOURCE SCRIPTS

(57) Abstract

In BAAN software, a user defined script is created by copying a standard script to a custom directory and making the required changes to the standard script so as to obtain a custom script. The problem with this method is that each time the standard script is patched, the custom script will need to be changed manually with the same fixes. The invention, in the form of the Qkey preprocessor, provides a solution to this problem. First, the user creates a custom script containing only the user's add-ons and modifications to the standard script, whereby preprocessor directives are used to call or override code contained in the standard script. Second, the Qkey preprocessor uses the directives to automatically add to the custom script the BAAN software instructions necessary to make the appropriate calls to and to dynamically load the standard script. Finally, the user script is compiled.



*(Referred to in PCT Gazette No. 17/2000, Section II)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

PREPROCESSOR FOR ENCAPSULATING SOFTWARE SOURCE SCRIPTS

FIELD OF THE INVENTION

The present invention relates to a software program for preprocessing source scripts of existing software.

BACKGROUND OF THE INVENTION

Software has been developed by a company called Baan. One application of this software is in enterprise resource planning ("ERP"). Baan software contains source scripts. The source scripts written by the end user will be derived from a Baan standard object. This derivation is dynamic so, if the Baan standard object changes (i.e., through a patch), the current user object will automatically receive these changes. This is known as "encapsulating" the standard object so changes to it do not necessarily require changes to the additional code added by the end user.

SUMMARY OF THE INVENTION

Generally described, the present invention provides in a first embodiment an apparatus for preprocessing existing source scripts, comprising: (a) a computer having a memory storage device and a microprocessor; (b) an operating system stored in said memory storage device; (c) means for viewing an object; and, (d) means for permitting the overriding of the associated events of said object.

Accordingly, it is an object of the present invention to provide a preprocessor for source scripts without having the source code available.

Other objects, features, and advantages of the present invention will become apparent upon reading the following detailed description of embodiments of the invention, when taken in conjunction with the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The various features and advantages of the invention will be apparent from the attached drawings, in which like reference characters designate the same or similar parts throughout the figures, and in which:

Fig. 1a is a compilation time process flow diagram for Standard Baan,

Fig. 1b is a compilation time process flow diagram for ENGEN Enhanced,

Fig. 2a is a run time process flow diagram for Standard Baan,

Fig. 2b is a run time process flow diagram for ENGEN Enhanced,

Fig. 3 is a logic flow diagram for ENGEN,

Fig. 4 is a visual display relating to installation in UNIX, and

Fig. 5 is a visual screen display relating to user configuration in UNIX.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In general, the present invention provides software for preprocessing source scripts. Details of the invention are set forth in the following documents attached hereto and incorporated herein: (1) User and Technical Guide, and (2) the code in the attached QSET for Baan Tools 6.1 and later and (3) a brochure entitled "Unlock the full potential of your Baan ERP system with QKEY...".

The user of the present invention can preprocess source scripts without having the source code available. This is possible by allowing the user to view each Baan standard session as an object, and by permitting the overriding or extension of the object's associated events (or "public events" in object oriented programming terminology). The result is that a user can create customized 4GL objects that

contain only the user's new code so that when a Baan patch or version release control ("VRC") is installed the changes can be reused automatically. In most cases, there is no need to recompile customizations. Even in installations that have already purchased Baan's source code, these features can save considerable time and money.

Figs. 1a and 1b provide a comparison of the compilation time processes of Standard Baan vs. ENGEN Enhanced, Figs. 2a and 2b provide a comparison of the run time processes of Standard Baan vs. ENGEN Enhanced, and Fig. 3 provides and overview of the logic process of ENGEN. In relation to Fig. 2a, it should be noted that all objects share a common memory area as they are a single process. All external variables, including table buffers are shared. In relation to Fig. 2b, it should be noted that as with the Baan standard process, all objects share a common memory area as they are a single process. All external variables, including table buffers are shared. 4GL parent is dynamically loaded so any patches are automatically included at runtime (no recompilation is necessary). Figs. 4 and 5 are described in the User and Technical Guide.

Among the advantages of the present invention are that there is significantly lowered programming costs, ease of implementation and reduced development time.

Although only a few exemplary embodiments of this invention have been described in detail above, those skilled in the art will readily appreciate that many modifications are possible in the exemplary embodiments without materially departing from the novel teachings and advantages of this invention. Accordingly, all such modifications are intended to be included within the scope of this invention as defined in the following claims.

It will be noted that the User and Technical Guide, and the code, and the brochure included herein comprise copyrighted matter owned by Quality Consultants, Inc. Also, QKEY and QSET are trademarks of Quality Consultants, Inc. Baan is a registered trademark of the Bann Company. All other trademarks or registered trademarks included herein are the property of their respective owners. It should further be noted that any patents, applications or publications referred to herein are incorporated by reference in their entirety.

QKEY
Encapsulated Generator for Baan

Version 2.10

User and Technical Guide

April 12, 1998

Table of Contents

Legal Notice....4
Introduction....4
Requirements....4
Supported Environments....4
Release History....2
Version 2.10....4
Version 2.00....2
Version 1.22....4
Version 1.21....4
Version 1.20....4
Version 1.10....4
Upgrading from v 1.x to v 2.x....4
Upgrading from v 1.10 to v 1.20....4
Installation in Unix....4
User Configuration in Unix....4
Concepts....4
Objects as DLLs....4
The Child Object....4
The Derivation Process....4
Using QKEY....4
Debugging....4
Reference....4
}#parent [object] 4
|#inhibit....4
|#call....4
|#pobj....4
|#end....4
Benefits....4
Limitations....4
Examples....4

Legal Notice

Introduction

The QCI Key Developer (QKEY) for Baan is designed to help the end user develop add-on products and enhancements for the Baan system. This tool works as a preprocessor for Baan source scripts. The source scripts written by the end user will be derived from a Baan standard object. This derivation is dynamic so, if the Baan standard object changes (i.e. through a patch), the current user object will automatic receive these changes. We call this "encapsulating" the Baan standard object so changes to it do not necessarily require changes to the additional code added by the end user.

QKEY is very easy to install and use. There are some limitations (see Limitations section), however, it does not limit any current development capability within Baan. QKEY may be used with or without standard source code.

Requirements

QKEY 2.0 and later, is now written in C so will be more generally available. However, because of the nature of compiled languages, it may not be available on all platforms.

To be able to use QKEY you must be using Baan Tools version 6.1 or later. Earlier versions did not use dynamic link libraries for 4GL objects. This means that Baan (Triton) 3.1a and later can take advantage of QKEY.

Note that the QKEY 1.x versions were written in Perl (4.x and 5.x) and were held to a limited release. However, these versions were available on all Unix platforms, as the only requirement was that Perl 4.0 or later be available. If you need a copy of one of these versions they may or may not be available. This will require a special agreement between your company and QCI as these are distributed in source code form and trademark secrets can be fully viewed.

Supported Environments

Version	Operating System	Development	Runtime Library	Availability
1.11	All Unix	Yes	Yes	Controlled
1.20	All Unix	Yes	Yes	Controlled
1.21/1.22	All Unix	Yes	Yes	Special
	Windows NT	No	Yes	
2.00	HP-UX 10.x and later	Yes	Yes	General
	All other Unix	No	Yes	
	Windows NT	No	Yes	
2.10	HP-UX 10.x and later	Yes	Yes	General
	All other Unix	No	Yes	
	Windows NT	Yes	Yes	

Release History

Version 2.10

Version 2.10 is a completed port to the Windows NT environment. QKEY is now available on Windows NT and Unix systems (see the section on *Supported Environments* for an accurate list).

Version 2.00

QKEY is fully rewritten in the "C" language. This improves performance and also allows for more manageable distribution and licensing of the product.

Change this product name from ENGEN to QKEY for public release.

Version 1.22

Began porting to Windows NT environment of Baan. This version fixes some small bugs in the search path scans as Windows NT includes driver letters, which also have a colon (:). The colon was previously used only as a path separator character. This fix requires an update to both the QKEY generator and the Baan library tccomqcidll1.

Note: This version was never released.

Version 1.21

This is a fix release to adjust for some changes in Baan IVc. This version of Baan has changed on Unix to more closely match the Windows NT version. Message output had to be redirected to a different location so Baan would pick it up in the display window.

Cleaned up some warning messages that were generated by the Baan compiler when using the "optimized" call method.

Fixed a problem resulting from the use of the string "${BSE}" in the file paths for Package VRCs. The Baan IVc release defaults to using this, however, this could cause problems in prior releases as the paths are not properly searched.

Note: This version was never released.

Version 1.20

In addition to some minor bug fixes, QKEY 1.20 removes the restriction that an QKEY 4GL object must be derived directly from a standard 4GL object. With this additional capability QKEY can be used at multiple levels in a single Package VRC structure. For example, if you are making enhancements to customer maintenance (tccom1101m000) which comes in Baan localized VRC B40L_b2_glo0 and have two VRCs derived from this, let's say B40C_b2_dev1 and then B40C_b2_dev2 derived from that. You can have QKEY code in both dev1 and dev2 without conflict. In both cases, QKEY will derive from the Baan localized object at the glo0 level. This added functionality requires that all QKEY code created in prior versions of QKEY be recompiled as QKEY implements this feature by adding a special external function to each of your QKEY 4GL objects. By doing this the routine to load the parent DLL object can detect when it attempts to load a QKEY object and bypass it. Note: Source created with QKEY 1.1 and earlier needs to be recompiled only if you use QKEY in more than a single level in your VRC tree.

Added "function call optimization". This is optional and still in testing. Basically function call optimization gets all the function identifiers for the parent event handling code when the

program starts. Then, when the parent must be called, the QKEY 4GL object can call the parent immediately without first having to lookup the function id in the parent. Function call optimization does add some overhead to the start up code (before.program) section so it is optionally available. Please contact a representative from QCI to learn how to activate this feature.

Fixed a bug where the "default:" tag for the case statement was considered a section name.

Added identifying marks for all QKEY messages to more clearly delineate these from the Baan standard Generator and Compilers.

Added a new utility script called *endiff* to be used to make source file comparisons. This is similar to *envi* and *enview* and will temporarily strip out QKEY added code so the base source script can be compared with another source script. In other words the script commands added by QKEY between |#pobj and |#end are removed so they don't interfere with the comparison. To learn about how to use this utility script, see the section *User Configuration*.

Version 1.10

Initial publicly available release.

Upgrading from v 1.x to v 2.x

Again, follow all instructions for *Installation*. There is again a new version of the dll library, be sure to install this as well. Use the same instructions given from *Upgrading from v 1.10 to v 1.20* below for determining all the places you may have installed a previous copy of the dll.

Upgrading from v 1.10 to v 1.20

Follow all the instructions for *Installation*. Do not leave out the final step – loading the support dll library – even though you already have a copy installed. There is a new version of the library in version 1.20. Be sure to load this library to every VRC the previous one was loaded in. A good way to verify you have gotten them all is to use the report Print Program/Library Scripts and use the sort for Program/Library then VRC. On the report, select package tc, all VRCs, and the library "comqcidll1". The report will show you all the installed VRCs. Also remember to check all your Baan environments if you have more than one.

Installation in Unix

The primary preprocessor program (the core of QKEY) is a single program which will be installed in the place of the Baan standard 4GL generator program. The 4GL generator will be renamed so it can still be used by QKEY. The installation will also install the command scripts *envi* and *endiff* which can be used to strip out QKEY generated code before using the vi and diff commands (see *User Configuration*).

1. Copy the distribution .tar (qkey-v2.00.tar or qkey-v2.00.tar.gz) file from the diskette to your host system. Use a program similar to ftp. If you are using ftp, be sure to choose "binary" transfer mode.

Example.

In this example, the name of the system where QKEY will be installed is "pluto" and the directory which will hold the installation files is called "installdir" in the bsp user's home directory.

ftp pluto
bsp
<enter password>
binary
cd installdir
put qkey-v2.00.tar or *put qkey-v2.00.tar.gz*
quit

2. Login to the host as "bsp" and be sure BSE is correctly set for your Baan environment. If not, type "BSE=<bse directory>; export BSE".

Example.

If the Baan Shell Environment is /baan/baan4/bse, you would use this.

BSE=/baan/baan4/bse
export BSE

3. Change to the directory containing the QKEY installation file.

4. If you have received a compressed tar file (the file name ends with ".gz" extension) then decompress the file using "gzip -d qkey-v2.00.tar.gz". This will change the file name to "qkey-v2.00.tar".

5. Expand the .tar file by typing "tar xvof qkey-v2.00.tar"

6. Type "sh ./install.qkey". This will copy the correct qkey command file to $BSE/bin, rename std_gen6.1 and create a symbolic link to qkey. The utility edit scripts, envi, enview, and endiff will all be installed at the same time. The install.qkey script may be executed multiple times without causing problems.

7. Import the QKEY DLL into each tc package VRC where you will be using QKEY. Use Import Data Dictionary to load the library source and object; the directory is "<installpath>/tcdll" and select "load to different Package VRC" and enter your PVRC. You can load this in a single higher level VRC that all other custom VRCs derive from as it will be visible in any lower level VRC. In Baan IVa or earlier, run "patch objects after error solving" to update the object header so it passes the license check.

SEE FIGURE 4

8. At your discretion, remove all the installation files (you may want to keep them around as they are small and if you create other VRC structures you may need to import the DLL again). Note, a Microsoft Word formatted document is also included in the distribution which contains this manual (qkey2.00.doc).

REMEMBER THIS! Any time you upgrade the porting set in Baan, you will need to reinstall QKEY. If you want, you can make the changes by hand to reactivate QKEY; as follows:

```
cd $BSE/bin
mv std_gen6.1.real std_gen6.1.real-# Save a copy of the previous version like Baan does
mv std_gen6.1 std_gen6.1.real
ln –s qkey std_gen6.1
```

User Configuration in Unix

Unless you want to have your developers access the *envi* or *endiff* scripts, you will not need to change any user configuration entries in Baan.

The *envi* script is available as a preprocessor for the source code prior to starting "vi" or "view" – the standard Unix editors for Baan scripts. If you want to use a different editor, copy *envi* to a new name and change the script to suit your purposes.

This script will strip out any code between the "|#pobj" and "|#end" directives before starting vi/view. These directives delineate the code automatically created by the QKEY process. By stripping the automatic code out, you may find it easier to edit and/or view your scripts. Nothing will be done to normal (non-QKEY) scripts as they will not contain |#pobj/|#end directives.

To activate *envi* or *enview* for a user, change the Developers Data for that user. The fields to change are "Editor Read-Only Command", which should be set to "enview", and "Editor Read/Write Command" which should be set to "envi". You can chose to only set one of the commands to the en* version (e.g. envi instead of vi).

The *endiff* command is intend to help make source comparisons easier to read. The QKEY code will be stripped prior to the *diff* command being executed. This means that only the code you created is actually compared and this is the only significant code – QKEY generated code is only important for calls to the parent object to keep the entire application program functioning as expected. To use *endiff*, set the "Difference Command" field to "endiff" instead of "diff" (not shown in example).

SEE FIGURE 5

Remember, the debugger will always show all the code between the |#pobj/|#end directives so the line numbers will still be correct unless you edit the script and don't recompile it.

Caution! Using "envi" will make it easier to edit your scripts, however, you may find it difficult to find the lines numbers reported by the Baan compiler. The compiler works with the source with all the code generated by QKEY as well as your source so line numbers will appear different to it than to the editor **if and only if** the generated lines are stripped (as with envi).

Hint. I use "envi" for my Editor Read/Write Command and "view" still for my Read Only Command. This allows me to see the source as the compiler does and get to a specific line number by selecting View from the maintenance session instead of Edit. You can make changes and save them with "view" by using the ":w!" or ":x!" commands (these override the Read Only value set in vi).

Concepts

Objects as DLLs

Unlike in previous version, in Tools 6.1 all Baan 4GL programs are really compiled into DLL objects. These objects are dynamically loaded and processed using the precompiled standard program. This is different than prior versions of Tools, where the source for the standard program was merged with the 3GL generated source for the custom 4GL script before compiling. Understanding this concept is crucial to understanding QKEY. In essence, all Baan 4GL objects are just function libraries holding the variables and code for the sections/subsections and functions of the script.

Since these objects are DLL's, the Baan standard object can be loaded dynamically as a DLL to your script. The only problem at this point is generating all the external calls at the right sections and subsections so the original functionality remains intact. The solution is QKEY, which generates all the necessary calls by retrieving all exported DLL functions that match section/subsection function names.

This opens up a number of possibilities in coding as it becomes more like traditional OOP. The "parent" object can be viewed as an OOP object, holding the inherited data (the external variables and tables) and methods (the section and subsection code). You can then add your own data and can choose to call the ancestor's methods or override (not use) those methods. For the standard program to recognize the functions in the session object, each section/subsection function must be exported by the object. Thus, all sections and subsections used by the parent object must be declared in any derived objects – QKEY transparently manages this for you. The variables (and tables) don't have to be declared in the initial object loaded by the standard program, they just need to be made visible by one of the DLLs loaded at the time they are resolved to actual addresses – sometime after the "before.program" section executes.

The Child Object

If your enhancements need to refer to tables or variables declared by the Baan standard program, they will need to be declared again (as "extern") in your script. The Baan compiler will be working only with your source and will not be checking any parent object for declarations. This works because the "extern" variables all point to the same memory location within the single bshell process.

There is some special code that must be added to the before.program section. In this section, the parent object is located and loaded. To locate the parent, there is a function that will search for the actual name of the object above the current object's location in the Package VRC tree. It does the search by first locating the object using Baan's pathname() function. Then, it opens the $BSE/lib/fd.6.1.<pacc> file and finds the PVRC search path for the object, scans the path until it passes the original object and then locates the next available one. The final object found is the one returned by the function – this gives us the object immediately "above" in the VRC search path. It has to be done this way, because to use just the object name in the load_dll() function, would cause a loop as the program tries to load itself and bshell will quickly have a stack error.

In summary, QKEY takes the steps necessary to determine what sections and subsections are used by the parent object, creates the necessary calls to these exported functions at the appropriate locations in your script, and adds code necessary to dynamic search for and load the parent object. If the new script contains a section that was previously used, QKEY adds to the code you have written (see |#inhibit and |#call for ways to control this). If there is a section that you haven't used, then the section is added automatically.

The commands for QKEY are always prefixed with "|#" which is interpreted by the Baan standard generator and compiler as comments. In this way these commands can be left in the code without confusing the compilation process. All code added by QKEY is placed between a pair of directives, |#pobj and |#end. **Do not attempt to place anything between these directives.** Anything placed between these will be striped the next time QKEY (or envi/enview) runs so it can regenerate the new code based on the current source.

The Derivation Process

Here's a brief description of how QKEY changes the development process and execution of Baan objects. In this description we'll be working with the sales detail line object and script (session tdsls4102s000), the Baan standard code is in VRC tdB40_a and the custom code is in tdB40C_a_cust. The original object and source are physically stored as:

object: $BSE/application/tdB40_a/otdsls/osls4102
source: $BSE/application/tdB40_a/ptdsls/psls41020

Under the normal (non-QKEY) developed method you would see this:

Copy source (using copy to current PVRC from script session) to custom PVRC; file $BSE/application/tdB40C_a_cust/ptdsls/psls41020 created.

Edit source as needed.

Compile source – new object is created as $BSE/application/tdB40C_a_cust/otdsls/osls4102 and Baan no longer uses (or cares about) the original object in $BSE/application/tdB40_a.

In time, when the standard source and object are patched, the new source file will need to be changed **by hand** with the same fixes. This same problem will occur when moving to a new VRC tree and there is a new version of the source file.

Using the enhanced QKEY method:

Create new, empty script, in custom PVRC (td40C_a_cust). Initially this is empty. Note – The source file in $BSE/application/tdB40_a/ptdsls is not needed with QKEY.

Add to **empty** source as desired – only adding code that you want. |#parent must be placed in the declaration section. QKEY will add any necessary sections/sub-sections including the before.program section if you don't already have it (or add to the one you do have).

Compile source – derived object created in td40C_a_cust. When Baan runs this object, the first thing it does is search for the parent object ($BSE/application/tdB40_a/otdsls/osls4102) and load it. Later, calls will be made into the parent object as each section/sub-section is executed in the custom version of the object.

In time, when the original object is patched, at most the custom object will just need to be recompiled. However, in most circumstances, the custom object can be left alone and the patches will immediately be used.

Using QKEY

Once installed, the QKEY preprocessor is transparent. It replaces the Baan standard generator (std_gen6.1) so Baan will automatically run QKEY instead of the generator. If QKEY finds no errors and was able to successful process the source file, it will call the original generator to finish the process. Here is a list of steps you need to take to use this capability in a new session script.

1. Start a new script. If you try to copy a Baan standard script entry (and you don't have source code), the tools record will be copied, but there will still be no source file. You will not be abie to edit the file as Baan requires at least an empty file to exist. In this case you will need to create an empty file in the appropriate directory ($BSE/application/....). An easier way is to call up the Baan script entry in Tools and insert a new record; accept all the defaults as they should come from the entry you just brought up. Baan will then create a new script file for you.

   Remember: If you use CTRL-X in Scripts to copy to current PVRC, the script record is copied but there's no source so that's not copied. Then when you try to use "edit source" command, there is still no source file so Baan just gives an error message. You then must create the empty file to bypass this error.

   Note: If you have Baan source and you still wish to use QKEY (there are several reasons to), remember that Baan will copy the source script to your new VRC when you choose either "copy" or "insert". There is no way around this. Just go ahead and let it make the copy, start the editor, and delete all the lines. You can then start with a clean source file and add just your enhancements.

2. Add the supplied DLL – `tccomqcidlli` - to the list of libraries for the script. (The find_parent function could have been added to each source file by QKEY but this was placed in a common DLL so the code only exists once; any fixes to find_parent can be made once).

   Note for those upgrading to Version 1.20 and later from earlier versions. The updated "find_parent" function is compatible with all previous compiled versions of QKEY and there is no need to recompile your source. However, to be able to work in 2 or more VRC levels with QKEY you still will need to recompile as the old objects don't have the identifying external function.

3. To create a derived object, add the "|#parent" directive to the declaration section of the new source code. The QKEY generator will automatically create all the code need to call the parent object and change the before.program section to load the parent object dynamically. All code added by QKEY is placed between the directive pair "|#pobj" and "|#end" (see Reference section for warnings about placing code between these directives).

   Example.
   declaration:
   table ttiitm001
   table ttdsls041

   |#parent

   If the "|#parent" command is left out, QKEY doesn't do anything to the source **except** strip any "|#pobj" and "|#end" pairs from the code. If no derivation directives had previously been used then the source file is left unchanged.

4. If the you do not want the code for a specific section/subsection in the parent called, the "|#inhibit" directive can be placed in that specific section/sub-section of the new source file.

   Example.

tdsls042.oqua:
check.input:
... your new code ....

|#inhibit

The parent object's check.input code will NOT be called for the tdsls042.oqua field.

5. If you want the code for a specific section/subsection in the parent called before the end of the section/subsection, place the "|#call" directive at the point you want the parent object called.

Example.
tdsls042.oqua:
check.input:
... your new code ....
|#call →
...some more of your code....

The parent object's check.input code will be called now.

Debugging

All code you create will be fully accessible in the Baan Tools debugger. The QKEY preprocessor leaves the generated code in your source file between the |#pobj/|#end directives so the debugger has the correct line number information. Notice that when the parent object is called, the debugger will not follow code execution into it unless it was also compiled with debugging. If you do not have the original source code you will not be able to debug into the parent object.

Reference

Following is a list of the directives understood by the QKEY preprocessor for controlling the derivation process.

|#parent [object]

This statement may appear only once in the source script and it must be in the declaration section. This activates the QKEY preprocessor on the script. Optionally, an object name can be specified so a script can derive from an object that does not match the script name (see example 2).

Example 1.
declaration:

```
... ...
|#parent
... ...
```

In this example, QKEY will seek out the parent object that matches the source file name. For instance, if the current source name is ptiitm01010 then the object name will be otiitm0101.

Example 2.
declaration:

```
... ...
|#parent otdsls4102
... ...
```

Here, QKEY will seek out object otdsls4102, no matter what the source file name is. The object found by QKEY will always be in a higher level VRC, even if there is an object in the current VRC level called otdsls4102.

|#inhibit

This directive can be used once per section (only if the section actually can have code, such as before.program) or subsection to keep the standard parent code from being executed. This will probably be used rarely as the larger use for QKEY is to add functionality to standard Baan sessions.

|#inhibit and |#call are mutually exclusive and should not be used in the same section/subsection.

Example.

```
field.tdsls042.oqua:
when.field.changes:
    ... ...
    tot.wght = tot.wght - prev.wght + (tdsls042.oqua * tiitm001.wght)
    ... ...
    |#inhibit
```

Here, the parent code for field tdsls4102.oqua, subsection when.field.changes will not be called. All other sections and subsections are uneffected. In this example, |#inhibit cannot be used immediately following the "field.tdsls4102.oqua" section name as code can not be legal placed at this point.

|#call

This directive can be used once per section or subsection to specify when the parent object code is called. Normally the parent object is called at the end of the section/subsection so any new code you add will be executed first.

|#inhibit and |#call are mutually exclusive and should not be used in the same section/subsection.

Example.

on.main.table:
before.delete:
... delete unrelated new rows here ...
|#call
... delete additional related rows here ...

The parent object routine for before.delete will be called after executing the first set of lines and before the second set of lines.

|#pobj

Delineates the start of code automatically added by QKEY. **Do not place code after this directive and before the succeeding |#end directive.** Code placed between |#pobj and |#end will be removed every time QKEY runs to strip what was generated on the previous run.

Caution! Do not add or remove |#pobj statements. It is best to use the QKEY (or envi) preprocessor to manipulate these. To remove all the directive pairs you can simply remove the |#parent directive from your script and recompile it.

Example.

```
|#parent
|#pobj added by QKEY 2.00
long   _pobj_dll_id
long   _pobj_func_id
long   _pobj_err
string _pobj_path(255)
|#end add by QKEY
```

|#end

Delineates the end of code automatically added by QKEY. **Do not place code before this directive and after the preceding |#pobj directive.** Code placed between |#pobj and |#end will be removed every time QKEY runs to strip what was generated on the previous run.

Caution! Do not add or remove |#end statements. It is best to use the QKEY (or envi) preprocessor to manipulate these. To remove all the directive pairs you can simply remove the |#parent directiven from your script and recompile it.

(see |#pobj for example)

Benefits

1. The new script only needs to contain the enhancements to the original code.

2. The new object **dynamically** loads the parent, thus, if the parent is changed the updated code is executed by the child. This could be helpful when upgrading to a new version of Baan. The way the program is created with QKEY, the custom object does not even need to be recompiled as long as the parent object has the same sections and subsections.

3. In the case that the sections and subsections in the parent have changed, the derived object only needs to be recompiled (QKEY takes care of adding the necessary code to the child script).

4. You need not have source code to create complex customizations to a standard session.

Limitations

There are some limitations with QKEY, but these limitations do **not** eliminate any current Baan functionality.

1. QKEY cannot change any current standard code (3GL functions). You can only add code around standard Baan routines or choose to not execute a standard routine. For example, in a when.field.changes subsection, which is part of a field section, you can choose to execute code before or after the Baan code for when.field.changes (using |#call). You can also choose to not execute Baan's standard code for this subsection (using |#inhibit), however, you **cannot** change what happens in Baan's standard code for this subsection.

2. QKEY cannot be used with 3GL code or report objects as there are few sections (none in 3GL code) that you can take advantage of. It has been designed to work with code tied to forms. In addition, std_gen6.1 is only executed for 4GL scripts; this is the only time QKEY will be called as it replaces the std_gen6.1 binary.

3. Limitation 3 removed in QKEY version 1.20! QKEY will automatically search for the real parent object in the VRC path. Any objects with the same name and generated by QKEY will be bypassed automatically in this search.

Examples

Here is some sample code for changes made to the sales order detail line display session (tdsls4503s000). Notice that QKEY was used to add calculations and new variables for the form.

Example 1.

Here is the code as written by the programmer.

```
|*********************************************************************
|* tdsls4503  0  VRC B40C b2 iac
|* Display line items during order entry
|* bsp
|* 05-22-97 [15:11]
|*********************************************************************
|* Script Type: 123
|*********************************************************************
|****************************** DECLARATION SECTION ****************
declaration:
       table  ttdltc001     | Lot information
       table  ttdsls041     | Sales order detail

       extern domain tdltc.clot   sls.lot
       extern domain tcamnt       l.neta   | Net price for line

       |#parent

|****************************** PROGRAM      SECTION ****************
|****************************** ZOOM FROM    SECTION ****************
|****************************** FORM          SECTION ***************
|****************************** CHOICE       SECTION ****************
|****************************** FIELD        SECTION ****************
field.sls.lot:
before.display:
       | Run number is the first 4 characters from the lot (if
specified)
       if tdsls041.lsel = tclsel.any then
              sls.lot = ""
              tdltc001.inf1 = ""
       else
              sls.lot = shift1$(tdsls041.clot)
              | Get the lot record for the selection code information
              select tdltc001.*
              from   tdltc001
              where  tdltc001._index1 = { :tdsls041.cprj,
                            :tdsls041.item, :tdsls041.cntr,
:tdsls041.clot }
              selectdo
                     continue
              endselect
       endif

field.l.neta:
before.display:
       | Compute net amount for item (Extended / Qty Ordered)
       l.neta = tdsls041.amta / tdsls041.oqua

|****************************** MAIN TABLE   SECTION ****************
|****************************** FUNCTION     SECTION ****************
```

Example 2.

Here is the same code as shown in Example 1 after it has been processed by QKEY version 1.10 and compiled by Baan. QKEY 1.20 and later are similar though they use some defines and have the potential of creating a table of functions identifiers for faster execution. Function optimization is still in testing so may or may not be seen in your generated source.

```
|*****************************************************************
|* tdsls4503  0  VRC B40C b2 iac
|* Display line items during order entry
|* bsp
|* 05-22-97 [15:11]
|*****************************************************************
|* Script Type: 123
|*****************************************************************
|****************************** DECLARATION SECTION ***************
declaration:
       table  ttdltc001     | Lot information
       table  ttdsls041     | Sales order detail

       extern domain tdltc.clot   sls.lot
       extern domain tcamnt       l.neta   | Net price for line

       |#parent
       |#pobj  --ADDED BY QKEY
       long   _pobj_dll_id
       long   _pobj_func_id
       long   _pobj_err
       string _pobj_path(255)
       |#end  --ADDED BY QKEY

|****************************** PROGRAM      SECTION ***************
|****************************** ZOOM FROM    SECTION ***************
|****************************** FORM          SECTION **************
|****************************** CHOICE       SECTION ***************
|****************************** FIELD        SECTION ***************
field.sls.lot:
before.display:
       | Run number is the first 4 characters from the lot (if
specified)
       if tdsls041.lsel = tclsel.any then
              sls.lot = ""
              tdltc001.infl = ""
       else
              sls.lot = shiftl$(tdsls041.clot)
              | Get the lot record for the selection code information
              select tdltc001.*
              from   tdltc001
              where  tdltc001._index1 = { :tdsls041.cprj,
                            :tdsls041.item, :tdsls041.cntr,
:tdsls041.clot }
              selectdo
                     continue
              endselect
       endif

field.l.neta:
before.display:
       | Compute net amount for item (Extended / Qty Ordered)
       l.neta = tdsls041.amta / tdsls041.oqua
       |#pobj  --ADDED BY QKEY



|*****************************************************************
```

```
|* Autogenerated sections/subsections for all remaining exported
|* functions from parent object.
|*********************************************************************

before.program:
        | This must be executed first; it loads the parent object
        | dll so it's routines can be called.
        | The "find_parent_obj" is an exported function that is in
        | the tccomqcidll1 library.
       if (find_parent_obj("otdsls4503", "otdsls4503", _pobj_path) <
0) then
         message("Unable to find parent object dll!")
         stop()
       endif

       _pobj_dll_id = load_dll(_pobj_path,0)
       if (_pobj_dll_id <= 0) then
         message("Unable to load parent object dll!")
         stop()
       endif
       _pobj_func_id = get_function(_pobj_dll_id, "before.program")
       _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

field.itm.dsca:
before.display:
       _pobj_func_id = get_function(_pobj_dll_id,
"before.display.itm.dsca")
       _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

field.tdsls041.disc:
before.display:
       _pobj_func_id = get_function(_pobj_dll_id,
                       "before.display.tdsls041.disc")
       _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

field.tdsls041.pono:
before.display:
       _pobj_func_id = get_function(_pobj_dll_id,
                       "before.display.tdsls041.pono")
       _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

form.1:
init.form:
       _pobj_func_id = get_function(_pobj_dll_id, "init.form.1")
       _pobj_err = exec_function(_pobj_dll_id, _pobj_func_id)

    |#end  --ADDED BY QKEY

|****************************** MAIN TABLE  SECTION ****************
|****************************** FUNCTION    SECTION ****************
```

```
/*****************************************************************************
  QSET (previously known as ENGEN) for Baan Tools 6.1 and later

  Author     : Kevin Brock
  Company    : Quality Consultants, Inc. (Process Technology Group)
  Created    : May, 1997 & March, 1998
  Tools Vers: 6.1

  Copyright (C) 1997,1998 Quality Consultants, Inc.
=============================================================================
Version     Date         Who  Comments
------- -------  ------ ---------------------------------------------------
        05/17/97 KRB    Initial version started
  2.0   03/28/98 KRB    Port ENGEN from Perl to C
*****************************************************************************/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pwd.h>
#include <ctype.h>

#include "misc.h"

#define COPYRIGHT "Copyright (C) 1997,1998 Quality Consultants, Inc."
#define VERSION   "2.0"
#define MYNAME    "ENGEN"

#define BAANTOOLSVER "6.1"

#define FILENAMESIZE    30
#define PATHSIZE        256
#define OBJNAMESIZE     100

#define EOS             '\0'
#define WHITESPACE      " \t\n\r"
#define nexttok(x)      (strtok(NULL, (x)))
#define strpartcmp(x,y) strncmp((x), (y), strlen(y))

struct pathst {
  char *pathcode;
  char *path;
  struct pathst *next;
  };

#define S_SECTION       1
#define S_SUBSECTION    2

struct sectionst {
  char *name;
  char *extfunc;        /* Exported function name */
  int  output_done;     /* =1 when output by |#call */
                        /* =2 when output at end of section/subsection */
  int  stype;           /* =1 for sections */
                        /* =2 for subsection */
  struct sectionst *firstsub;
```

```
  struct sectionst *next;
  struct sectionst *secp; /* This points back to the section structure */
                          /* Note: For sections, this points to itself */
  );

struct commentst {
  char *comment;
  struct commentst *next;
  );

/* Parameter settings */
int  optimize_function_calls = FALSE;

/* Environment */
char bse[PATHSIZE];
char bsetmp[PATHSIZE];
char user[20];
char pacc[20];
char bic_info[FILENAMESIZE];
char std_gen[FILENAMESIZE];

/* Internal global variables */
int  emit_holder = TRUE;  /* Time to output "|#pobj" directive       */
int  inhibit     = FALSE; /* Inhibit directive */
int  bp_found    = FALSE; /* before.program found in normal processing */
char current_section[OBJNAMESIZE];
char current_subsection[OBJNAMESIZE];
int  lineno        = 0;
int  numfuncs    = 0;
int  genobj      = 0;
int  lastlvl       = -1;   /* lastlvl is set by search_path and is the count
                    of the levels searched when looking through
                    the passed path name.  It would be used to
                    skip the same number of directory entries in
                    the path or similar path; generally to find
                    something further up the search path (parent
                    objects, etc). */

char command[MAXLINE]; /* temporary buffer; should be used quickly */

FILE *destfile;
FILE *srcfile;
char src_name[FILENAMESIZE];
char srcfullpath[PATHSIZE];
char src_line[MAXLINE];
char src_package[3], src_module[4], src_number[5];
char current_obj[FILENAMESIZE];
char parent_obj[FILENAMESIZE];

int save_stdout;
int save_stderr;
char baan4c_output[PATHSIZE];
char temp_output[PATHSIZE];

struct sectionst *s_list = NULL;  /* List of sections/subsections in parent */
struct sectionst *s_prev = NULL;  /* The previous section when searching */
struct sectionst *sb_prev = NULL; /* The previous subsection when searching */
```

```
struct pathst *path_list = NULL;  /* List of paths */
struct pathst *path_last = NULL;  /* End of path list */

struct commentst *cmt_list = NULL; /* List of comments */
struct commentst *cmt_last = NULL; /* End of comment list */

/* Prototypes as needed */
void end_program(int exitcode);

/*****************************************************************************
 * Support routines                                                          *
 *****************************************************************************/

void chop(char *s)
  {
  int l = strlen(s);

  if (l > 0)
    *(s + l - 1) = EOS;
  }

void rename_file(char *oldname, char *newname)
  {
  if (link(oldname, newname) == 0)
    {
    unlink(oldname);
    }
  }

static void engen_msg(char *msglevel, char *fmt, va_list ap)
{
  int   errno_save;
  char  buf[MAXLINE];
  char  fmtbuf[MAXLINE];

  errno_save = errno;
  if (msglevel == NULL)
    sprintf(fmtbuf, "%s: %s", MYNAME, fmt);
  else
    if (lineno)
      sprintf(fmtbuf, "%s (%d): %s: %s", MYNAME, lineno, msglevel, fmt);
    else
      sprintf(fmtbuf, "%s: %s: %s", MYNAME, msglevel, fmt);
  vsprintf(buf, fmtbuf, ap);
  strcat(buf, "\n");
  fputs(buf, stdout);
  return;
}

void fatal_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg("Fatal", fmt, ap);
  va_end(ap);
```

```
  end_program(1);
}

void error_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg("Error", fmt, ap);
  va_end(ap);
  return;
}

void warning_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg("Warning", fmt, ap);
  va_end(ap);
  return;
}

void info_msg(char *fmt, ...)
{
  va_list   ap;

  va_start(ap, fmt);
  engen_msg(NULL, fmt, ap);
  va_end(ap);
  return;
}

/*****************************************************************************
 * Section/Subsection list handling                                          *
 *****************************************************************************/

/* Note: These lists are kept in sorted order so output is sorted */

/* Look for the section name in the section link listed */
struct sectionst *find_section(char *section, int exact)
  {
    struct sectionst *s;
    int    cmp;

    s_prev = NULL;

    /* No list established yet! */
    if (s_list == NULL)
      return (NULL);

    for (s = s_list; s != NULL && (cmp = strcmp(s->name, section)) < 0;
         s_prev = s, s = s->next);
    if (exact)
      return (cmp == 0 ? s : NULL);

    return (s);
```

```
}

struct sectionst *find_subsection(struct sectionst *sectionp,
                                  char *subsection,
                                  int  exact)
{
  struct sectionst *sb;
  int    cmp;

  sb_prev = NULL;

  /* No section or subsection list */
  if (sectionp == NULL || sectionp->firstsub == NULL)
    return (NULL);

  /* Subsection name is null -- there will be none then */
  if (subsection[0] == EOS)
    return (NULL);

  for (sb = sectionp->firstsub;
       sb != NULL && (cmp = strcmp(sb->name, subsection)) < 0;
       sb_prev = sb, sb = sb->next);
  if (exact)
    return (cmp == 0 ? sb : NULL);

  return (sb);
}

int find_reference(struct sectionst **rets, char *section, char *subsection)
{
int ok;
struct sectionst *s;

s = find_section(section, TRUE);
if (s != NULL)
  {
  if (*subsection != EOS)
    {
    s = find_subsection(s, subsection, TRUE);
    }
  else
    {
    /* If a subsection name is not given but there a subsection list then
     this entry is an error as a subsection must be specified */
    if (s->firstsub != NULL)
    s = NULL;
    }
  }

*rets = s;
return (s != NULL);
}

/* Add a section and subsection to the parent's section table */
void add_reference(char *section, char *subsection, char *object)
{
  struct sectionst *s, *sb;
```

```
/* Locate the section and subsection within the list; the
   entry may only exist one time */
s = find_section(section, FALSE);
if (s != NULL && strcmp(s->name, section) == 0)
  sb = find_subsection(s, subsection, FALSE);
else
  s = sb = sb_prev = NULL;

/* Build a new section structure */
if (s == NULL)
  {
  if ((s = malloc(sizeof(struct sectionst))) == NULL)
    {
    fatal_msg("unable to allocate memory for section %s", section);
    return;
    }
  memset(s, 0, sizeof(struct sectionst));
  s->name = strdup(section);
  s->stype = S_SECTION;
  s->secp = s;

  /* If there's no subsection name then the object name goes with
     the section */
  if (*subsection == EOS)
    {
    s->extfunc = strdup(object);
    }

  if (s_prev == NULL)
    {
    /* Insert at beginning of list */
    s->next = s_list;
    s_list = s;
    }
  else
    {
    /* Add to current position in list */
    s->next = s_prev->next;
    s_prev->next = s;
    }
  }

/* Build a new subsection structure (if there is a subsection) */
if (*subsection != EOS &&
    (sb == NULL || strcmp(sb->name, subsection) != 0))
  {
  if ((sb = malloc(sizeof(struct sectionst))) == NULL)
    {
    fatal_msg("unable to allocate memory for subsection %s:%s", section,
          subsection);
    return;
    }
  memset(sb, 0, sizeof(struct sectionst));
  sb->name = strdup(subsection);
  sb->extfunc = strdup(object);
  sb->stype = S_SUBSECTION;
```

```
        sb->secp = s;

        if (sb_prev == NULL)
          {
          /* Insert at beginning of subsection list */
          sb->next = s->firstsub;
          s->firstsub = sb;
          }
        else
          {
          /* Add to current position in subsection list */
          sb->next = sb_prev->next;
          sb_prev->next = sb;
          }
        }
    }

void dispose_sections(void)
    {
    struct sectionst *s, *sb, *stemp;

    /* Free the memory for the sections/subsections list */
    for (s = s_list; s != NULL;)
      {
      for (sb = s->firstsub; sb != NULL;)
        {
        if (sb->name != NULL)
          free(sb->name);
        if (sb->extfunc != NULL)
          free(sb->extfunc);
        stemp = sb;
        sb = sb->next;
        free(stemp);
        }

      if (s->name != NULL)
        free(s->name);
      if (s->extfunc != NULL)
        free(s->extfunc);
      stemp = s;
      s = s->next;
      free(stemp);
      }

    s_list = NULL;
    s_prev = NULL;
    sb_prev = NULL;
    }

/* Call this to begin enuermating all sections and subsections */
/* Note: Only used entities are returned.  In other words, a   */
/*   section that has subsections is not returned by itself.   */
struct sectionst *enum_s;
struct sectionst *enum_sb;

struct sectionst *enum_sections(void)
    {
```

```
  enum_s = s_list;
  enum_sb = enum_s->firstsub;
  return (enum_sb == NULL ? enum_s : enum_sb);
  }

/* Call this to get next section or subsection in enumerate list */
struct sectionst *enum_next(void)
  {
  if (enum_sb != NULL)
    {
    enum_sb = enum_sb->next;
    if (enum_sb != NULL)
      return (enum_sb);
    }

  /* Move to next section */
  enum_s = enum_s->next;
  enum_sb = enum_s->firstsub;
  return (enum_sb == NULL ? enum_s : enum_sb);
  }

/*****************************************************************************
 * Comment handling                                                          *
 *****************************************************************************/

void add_comment(char *comment)
  {
  struct commentst *c;

  c = malloc(sizeof(struct commentst));
  c->comment = strdup(comment);
  c->next = NULL;

  if (cmt_list == NULL)
    {
    cmt_list = c;
    cmt_last = c;
    }
  else
    {
    cmt_last->next = c;
    cmt_last = c;
    }
  }

void dispose_comments(void)
  {
  struct commentst *c, *ctemp;

  for (c = cmt_list; c != NULL;)
    {
    free(c->comment);
    ctemp = c;
    c = c->next;
    free(ctemp);
    }
```

```
  cmt_list = NULL;
  cmt_last = NULL;
  }

/*****************************************************************************
 * Pathlist handling                                                         *
 *****************************************************************************/

void add_path(char *pathcode, char *path)
  {
  struct pathst *p;

  p = malloc(sizeof(struct pathst));
  p->pathcode = strdup(pathcode);
  p->path = strdup(path);
  p->next = NULL;

  if (path_list == NULL)
    {
    path_list = p;
    path_last = p;
    }
  else
    {
    path_last->next = p;
    path_last = p;
    }
  }

struct pathst *find_path(char *pathcode)
  {
  struct pathst *p;

  for (p = path_list;
       p != NULL && strcmp(p->pathcode, pathcode) != 0;
       p = p->next);

  return (p);
  }

void dispose_paths(void)
  {
  struct pathst *p, *ptemp;

  for (p = path_list; p != NULL;)
    {
    free(p->pathcode);
    free(p->path);
    ptemp = p;
    p = p->next;
    free(ptemp);
    }

  path_list = NULL;
  path_last = NULL;
  }
```

```
/*****************************************************************************
 * Object search and load routines                                           *
 *****************************************************************************/

/* is_obj_engen() function checks to see if the obj file an engen
   generated object (it includes the function "_engen_object_version") */
int is_obj_engen(char * filename)
  {
  FILE  *pfile;
  int   is_engen;
  char  linebuf[MAXLINE];

  is_engen = 0;

  /* Open a pipe from bic_info (Baan's tool to report information on
     objects) */
#ifdef WINNT
#else
  sprintf(command, "%s/bin/%s -e %s", bse, bic_info, filename);
  if ((pfile = popen(command, "r")) == NULL)
    error_msg("unable to start %s; aborting", bic_info);
#endif

  while (fgets(linebuf, MAXLINE, pfile) != NULL)
    {
    chop(linebuf);

    if (strstr(linebuf, "ERROR") != NULL)
      {
      error_msg("%s reported an error; aborting", bic_info);
      break;
      }

    if (strpartcmp(linebuf, "function extern ") == 0 &&
        strstr(linebuf, "_engen_object_version") != NULL)
      {
      info_msg("Skip %s obj: %s", locasestr(MYNAME), filename);
      is_engen = 1;
      break;
      }
    }

  fclose(pfile);

#ifdef WINNT
#endif

  return (is_engen);
  }

/* This searches the appropriate path as retrieved from the fd6.1.<pacc>
   file.  The first parameter is the Baan standard name (ptdsls4102).
   If skiplvl is non-zero then start searching just past that position
   in the path list (skip all prior entries).

   If skipengen is true then skip any engen objects (those containing
   the extern function "_engen_object_version").  Note: This will only
```

```
   occur if skiplvl is also set.

   Returns a pointer to the full path to the file (including the file
   name) -- null if the files was not found. */
char * search_path (char * baanname, int skiplvl, int skipengen)
  {
  static char buf[PATHSIZE];
  char       filename[FILENAMESIZE], code[2], package[3], module[4], rest[21];
  char       pathcode[4], path[PATHSIZE], *p, *dir, *s, *d;
  struct pathst *pp;

  skipengen = (skiplvl ? skipengen : 0);

  /* Split the Baan standard name into the proper directory/filename */
  strcpy(code, substr(baanname, 1, 1));
  strcpy(package, substr(baanname, 2, 3));
  strcpy(module, substr(baanname, 4, 6));
  strcpy(rest, substr(baanname, 7, 26));
  sprintf(filename, "/%s%s%s/%s%s%s", code, package, module,
        code, module, rest);

  /* Locate the path name to use in the loaded path name list */
  strcpy(pathcode, code);
  strcat(pathcode, package);
  if ((pp = find_path(pathcode)) == NULL)
    return (NULL); /* No valid path found! */
  strcpy(path, pp->path);

  /* Skip past directories if requested */
  p = path;
  lastlvl = 0;
  while (skiplvl > 0)
    {
    /* To support Windows NT path names, we must skip past at least the
       second character -- we can assume the path will be at least larger
       than 2 characters and the second one may be a legitimate ":" to
       identified the drive */
    if (*(p+1) == ':') p += 2;
    while (*p != EOS && *p != ':') p++;
    skiplvl --;
    lastlvl ++;
    }

  /* Look for file in path list */
  while (p != NULL)
    {
    lastlvl ++;
    dir = p;
    /* Must skip past drive letter under Windows NT (see comment above) */
    if (*(p+1) == ':') p += 2;
    p = strstr(p, ":");
    if (p != NULL)
      {
      *p = EOS;
      p++;
      }
```

```
    /* Expand ${BSE} to the actual contents of the bse */
    if (strstr(dir, "${BSE}") != NULL)
      {
      strcpy(buf, "");
      for (s = dir, d = buf; *s != EOS;)
        {
        /* In theory this should always be at the beginning of the string,
           this can handle other situations, however, we will always
           assume that there is only one ${BSE} literal */
        if (strpartcmp(s, "${BSE}") == 0)
          {
          strcat(d, bse);
          s += 6;
          strcat(d, s);
          break;
          }
        else
          {
          *d = *s;
          d++;
          s++;
          }
        }
      }
    else
      strcpy(buf, dir);

    strcat(buf, filename);
    if (access(buf, R_OK) == 0)
      {
      if (!skipengen || (skipengen && !is_obj_engen(buf)))
        return(buf);
      }
    }

  /* No file found! */
  return (NULL);
  } /* search_path */

int loadparent(char *pobj)
  {
  char      line[256];
  char  object[100], *t, *p, parts[5][25];
  char  key1[50], key2[50];
  FILE      *dllfile;
  int   ok = TRUE;

#ifdef WINNT
#else
  /* Open bic_info6.1 command on the parent object */
  sprintf(command, "%s/bin/%s -e %s 2>&1", bse, bic_info, pobj);
  if ((dllfile = popen(command, "r")) == NULL)
    {
    error_msg("unable to start %s; aborting", bic_info);
    return (FALSE);
    }
#endif
```

```
numfuncs = 0;

while (fgets(line, 256, dllfile) != NULL)
  {
  chop(line);

  if (strstr(line, "ERROR") != NULL)
    {
    error_msg("%s reported an error; aborting", bic_info);
    ok = FALSE;
    break;
    }

  if (strpartcmp(line, "function extern ") == 0)
    {
    /* Skip past first two tokens -- that is "function" and "extern" */
    t = strtok(line, WHITESPACE);
    t = nexttok(WHITESPACE);

    /* This is the name of the external function */
    strcpy(object, (t = nexttok(WHITESPACE "(")));

    /* Is the object name one we are interested in?  It must consist only
       of Upper/Lower/Numeric characters */
    for (p = t; *p != EOS && isalnum(*p); p++);
    if (*p != '.') continue;  /* name potion must end in period */

    /* Split the object name into components based on "." delimiter */
    /* This uses the original line buffer as it's no longer needed and
       keeps the "object" variable intact for later processing. */
    strcpy(parts[0], strtok(t, "."));
    strcpy(parts[1], nexttok("."));
    strcpy(parts[2], nexttok("."));
    strcpy(parts[3], nexttok("."));
    strcpy(parts[4], nexttok("\n"));  /* Remainder of object name */

    if (strcmp(parts[1], "choice") == 0 || strcmp(parts[1], "form") == 0)
      {
    sprintf(key1, "%s.%s.%s.%s", parts[1], parts[2],
              parts[3], parts[4]);
    sprintf(key2, "%s.%s", parts[0], parts[1]);
      }
    else if (strpartcmp(object, "init.field.") == 0 ||
          strpartcmp(object, "before.field.") == 0 ||
          strpartcmp(object, "after.field.") == 0 ||
          strpartcmp(object, "before.input.") == 0 ||
          strpartcmp(object, "after.input.") == 0 ||
          strpartcmp(object, "before.display.") == 0 ||
          strpartcmp(object, "after.display.") == 0 ||
          strpartcmp(object, "before.zoom.") == 0 ||
          strpartcmp(object, "after.zoom.") == 0 ||
          strpartcmp(object, "before.checks.") == 0 ||
          strpartcmp(object, "domain.error.") == 0 ||
          strpartcmp(object, "ref.input.") == 0 ||
          strpartcmp(object, "ref.display.") == 0 ||
          strpartcmp(object, "check.input.") == 0 ||
```

```
      strpartcmp(object, "on.input.") == 0)
  {
sprintf(key1, "field.%s.%s.%s", parts[2], parts[3], parts[4]);
sprintf(key2, "%s.%s", parts[0], parts[1]);
  }
else if (strcmp(parts[2], "zoom") == 0 && strcmp(parts[3], "from") == 0)
  {
  strcpy(key1, "zoom.from.");
  strcat(key1, parts[4]);
sprintf(key2, "%s.%s", parts[0], parts[1]);
  }
else if (strpartcmp(object, "when.field.changes") == 0)
  {
sprintf(key1, "field.%s.%s", parts[3], parts[4]);
  strcpy(key2, "when.field.changes");
  }
else if (strcmp(object, "after.update.db.commit") == 0 ||
    strcmp(object, "before.program") == 0 ||
    strcmp(object, "after.program") == 0 ||
    strcmp(object, "on.error") == 0)
{
strcpy(key1, object);
strcpy(key2, "");
}
else if (strpartcmp(object, "before.read") == 0 ||
      strpartcmp(object, "after.read") == 0 ||
      strpartcmp(object, "before.write") == 0 ||
      strpartcmp(object, "after.write") == 0 ||
      strpartcmp(object, "after.skip.write") == 0 ||
      strpartcmp(object, "before.rewrite") == 0 ||
      strpartcmp(object, "after.rewrite") == 0 ||
      strpartcmp(object, "after.skip.rewrite") == 0 ||
      strpartcmp(object, "before.delete") == 0 ||
      strpartcmp(object, "after.delete") == 0 ||
      strpartcmp(object, "after.skip.delete") == 0 ||
      strpartcmp(object, "read.view") == 0)
  {
  strcpy(key1, "main.table.io");
  strcpy(key2, object);
  }
else
  {
  /* Unknown exported function format */
  warning_msg("unknown exported function: %s", object);
  strcpy(object, ""); /* Invalidate the object as we don't know how
                         to handle it. */
  }

if (strcmp(object, "") != 0)
  {
  /* Strip any unnecessary periods from the end of the key strings */
  for (p = key1 + strlen(key1) - 1; *p == '.'; p--);
  p++; *p = EOS;
  for (p = key2 + strlen(key2) - 1; *p == '.'; p--);
  p++; *p = EOS;

  add_reference(key1, key2, object);
```

```
        numfuncs ++;
        }
      }
    }

  pclose(dllfile);

#ifdef WINNT
#endif

  return (ok);
  } /* loadparent */

/*****************************************************************************
 * Emitter section                                                           *
 *****************************************************************************/

void out_startpobj(void)
  {
  if (emit_holder)
    {
    fprintf(destfile, "\t|#pobj added by %s %s\n", MYNAME, VERSION);
    emit_holder = FALSE;
    }
  }

void out_endpobj(void)
  {
  if (! emit_holder)
    {
    fprintf(destfile, "\t|#end add by %s\n", MYNAME);
    emit_holder = TRUE;
    }
  }

char *get_defname(char *extfunc)
  {
  static char nbuf[80], *p;

  /* Map a function name with "."s to a #define name.  Also prepends
     "_f_" to the name */
  strcpy(nbuf, "_f_");
  strcat(nbuf, extfunc);
  for (p = nbuf; *p != EOS; p++) if (*p == '.') *p = '_';
  return (nbuf);
  }

void out_functiondefs(void)
  {
  struct sectionst *s;
  int    cnt;
  char   def[80];

  /* This should only be called if we're optimizing the function calls   */
  /* and in the declaration section.  It creates the definitions for the */
  /* array holding the extern function ids in the parent dll for use     */
  /* later in making the direct calls.                                   */
```

```
  if (! optimize_function_calls) return;

  out_startpobj();

  fprintf(destfile, "\tlong\t_pobj_func_ids(%d)\n", numfuncs);

  for (cnt = 0, s = enum_sections(); s != NULL; s = enum_next())
    {
    cnt++;
    strcpy(def, get_defname(s->extfunc));
    fprintf(destfile, "\t#define %s%s_pobj_func_ids(%d)\n", def,
            strrep('\t', 5 - (strlen(def)) / 8), cnt);
    }
  }

void out_dllload(void)
  {
  bp_found = 1;

  out_startpobj();

  fprintf(destfile, "\t| This must be executed first; it "
                    "loads the parent object\n");
  fprintf(destfile, "\t| dll so it's routines can be called.\n");
  fprintf(destfile, "\t| \"find_parent_obj\" is an exported function "
                    "that is in\n");
  fprintf(destfile, "\t| the tccomqcidlll library.\n");
  fprintf(destfile, "\tif (find_parent_obj(\"%s\", \"%s\", _pobj_path) "
                    "< 0) then\n", current_obj, parent_obj);
  fprintf(destfile, "\t  message(\"Unable to find parent object dll!\")\n");
  fprintf(destfile, "\t  end()\n");
  fprintf(destfile, "\tendif\n\n");
  fprintf(destfile, "\t_pobj_dll_id = load_dll(_pobj_path,0)\n");
  fprintf(destfile, "\tif (_pobj_dll_id <= 0) then\n");
  fprintf(destfile, "\t  message(\"Unable to load parent object dll!\")\n");
  fprintf(destfile, "\t  end()\n");
  fprintf(destfile, "\tendif\n");

  if (optimize_function_calls)
    {
    fprintf(destfile, "\n\t_pobj_get_fids()\n");
    }
  }

/* Output a parent call for a specific section/subsection. */
/* Use this if you have the pointer to the memory structure */
/* Returns 1 if succesful, 0 on failure */
int out_pcall_ref(struct sectionst *s, int callverb)
  {
  /* Have we already output the call information for this entity? */
  if (s->output_done)
    {
    if (callverb)
      {
      /* The "|#call" directive was specified, but we've already output */
      /* the call once, so it probably appears twice. */
```

```
      error_msg("|#call used more than once in %s:%s\n",
            s->secp->name, (s->stype == S_SUBSECTION ? s->name : ""));
      return (0);
      }

    if (s->output_done == 2)
      {
      error_msg("section %s:%s specified multiple times",
            s->secp->name, (s->stype == S_SUBSECTION ? s->name : ""));
      return (0);
      }

    /* Ignore this default generate request, the call to the parent */
    /* object was already generated with a |#call directive.        */
    s->output_done = 2;
    return (1);
    }

  if (callverb)
    s->output_done = 1;  /* Ok to see a default generate request */
  else
    s->output_done = 2;  /* Should never see this request again */

  /* Now, generate the code...unless the program has strictly inhibited this */
  if (! inhibit)
    {
    out_startpobj();

    if (optimize_function_calls)
      fprintf(destfile, "\t_pobj_exe0(%s)\n", get_defname(s->extfunc));
    else
      fprintf(destfile, "\t_pobj_exe1(\"%s\")\n", s->extfunc);
    }

  return (1);
  }

/* Output a parent call for a specific section/subsection. */
/* Use this if you have just the section/subsection names */
/* Returns 1 if succesful, 0 on failure */
int out_pcall(char *section, char *subsection, int callverb)
  {
  struct sectionst *s;

  /* Determine if the section/subsection has a parent related object */
  if (! find_reference(&s, section, subsection))
    {
    /* Parent didn't have a reference to this, consider this successful */
    return (1);
    }

  return (out_pcall_ref(s, callverb));
  }

/* Write out all inheritance calls for all remaining subsections for the */
/* passed section (these subsections are not defined in the current      */
/* source).                                                              */
```

```
void out_rest(char *section)
  {
  struct sectionst *s, *sb;
  int     first = TRUE;

  s = find_section(section, TRUE);
  if (s == NULL || s->firstsub == NULL)
    return;

  for (sb = s->firstsub; sb != NULL; sb = sb->next)
    {
    if (! sb->output_done)
      {
      if (first)
        {
        out_startpobj();
        fprintf(destfile, "
|*****************************************************************************
|* Autogenerated subsections for %s
|*****************************************************************************
\n",              section);
        first = FALSE;
        }

      fprintf(destfile, "%s:\n", sb->name);
      out_pcall_ref(sb, FALSE);
      fprintf(destfile, "\n");
      }
    }
  }

/* Write out all remaining inheritance calls */
void out_all()
  {
  struct sectionst *s, *lastsec;
  int     first = TRUE, sout;

  if (! bp_found && find_section("before.program", TRUE) == NULL)
    {
    out_startpobj();
    fprintf(destfile, "
|*****************************************************************************
|* Autogenerated sections/subsections for all remaining exported functions
|* from parent object.
|*
|* Note: before.program was not defined in this source and also is not
|*       in the parent, however, it must be added because this is where
|*       the parent dll object is loaded.
|*****************************************************************************
\n");
    fprintf(destfile, "before.program:\n");
    out_dllload();
    fprintf(destfile, "\n");
    first = FALSE;
    }

  for (lastsec = NULL, s = enum_sections(); s != NULL; s = enum_next())
```

```
    {
    if (s->output_done) continue;

    if (first)
      {
      out_startpobj();
      fprintf(destfile, "
|*****************************************************************************
|* Autogenerated sections/subsections for all remaining exported functions
|* from parent object.
|*****************************************************************************
\n");
      first = FALSE;
      }

    /* Output the section id line, if this is the first time */
    if (s->secp != lastsec)
      {
      fprintf(destfile, "%s:\n", s->secp->name);
      lastsec = s->secp;
      }

    if (s->stype == S_SECTION)
      {
      if (strcmp(s->name, "before.program")  == 0)
        out_dllload();
      }
    else
      {
      /* Subsection name */
      fprintf(destfile, "%s:\n", s->name);
      }
    out_pcall_ref(s, FALSE);
    fprintf(destfile, "\n");
    }
  }

/* Add the _engen_object_version function to the script.  This must      */
/* be the last item output to the new source.  The function section      */
/* must be the last one so we're either in it or can add it because none */
/* previously existed.                                                   */
void out_ident(void)
  {
  char *t;
  int  major, minor;
  char dummyc;

  out_startpobj();

  if (strcmp(current_section, "functions") != 0)
    {
    fprintf(destfile, "functions:\n");
    strcpy(current_section, "functions");
    }

  /* Figure out a numeric number for the current version */
  /* This assume the version string will be like: "#.#" */
```

```
  sscanf(VERSION, "%d%c%d", &major, &dummyc, &minor);

  fprintf(destfile, "function extern long _engen_object_version()\n");
  fprintf(destfile, "{\n");
  fprintf(destfile, "\treturn(%d%02d)\n", major, minor);
  fprintf(destfile, ")\n");

  out_endpobj();
  }

/* Only used if optimizing function calls.  This is the function called  */
/* in before.program which will set all the parent dll function ids.     */
void out_getfids(void)
  {
  struct sectionst *s;

  if (! optimize_function_calls) return;

  out_startpobj();

  if (strcmp(current_section, "functions") != 0)
    {
    fprintf(destfile, "functions:\n");
    strcpy(current_section, "functions");
    }

  fprintf(destfile, "function _pobj_get_fids()\n");
  fprintf(destfile, "{\n");

  for (s = enum_sections(); s != NULL; s = enum_next())
    {
    fprintf(destfile, "\t_pobj_func_id = _pobj_get(");
    if (strlen(s->extfunc) <= 26)
      fprintf(destfile, "\"%s\")\n", s->extfunc);
    else
      fprintf(destfile, "\n\t                    \"%s\")\n", s->extfunc);
    fprintf(destfile, "\t%s = _pobj_func_id\n", get_defname(s->extfunc));
    }
  fprintf(destfile, "}\n");
  }

void out_line(int skip_input)
  {
  struct commentst *c, *c2;

  out_endpobj();

  if (cmt_list != NULL)
    {
    for (c = cmt_list; c != NULL;)
      {
      fprintf(destfile, "%s\n", c->comment);

      c2 = c;
      c = c->next;
      free(c2->comment);
      free(c2);
```

```
      }
    cmt_list = NULL;
    cmt_last = NULL;
    }

  if (!skip_input)
    fprintf(destfile, "%s\n", src_line);
  }

/*****************************************************************************
 * Source input and main processing                                          *
 *****************************************************************************/

/* Determine if the string looks like a section/subsection line */
int looks_like_section(char *s)
  {
  for (; *s != EOS && isspace(*s); s++);   /* Skip leading whitespace */
  if (*s == EOS)
    return (FALSE);
  for (s++; *s != EOS && (isalnum(*s) || *s == '.'); s++);
  if (*s != ':')
    return (FALSE);
  for (s++; *s != EOS && isspace(*s); s++);  /* Verify remaining whitespace */
  if (*s != EOS)
    return (FALSE);

  return (TRUE);
  }

int is_section_name(char *s)
  {
  /* Determine if the passed name is the name of a Baan 4GL section name */

  /* Fixed session names */
  if (strcmp(s, "declaration") == 0 ||
      strcmp(s, "before.program") == 0 ||
      strcmp(s, "on.error") == 0 ||
      strcmp(s, "after.program") == 0 ||
      strcmp(s, "after.update.db.commit") == 0 ||
      strcmp(s, "functions") == 0 ||
      strcmp(s, "form.all") == 0 ||
      strcmp(s, "form.other") == 0 ||
      strcmp(s, "field.all") == 0 ||
      strcmp(s, "field.other") == 0 ||
      strcmp(s, "zoom.from.all") == 0 ||
      strcmp(s, "zoom.from.other") == 0 ||
      strcmp(s, "main.table.io") == 0)
     return (TRUE);

  /* Form session names */
  if (strpartcmp(s, "form.") == 0)
    {
    /* May only be followed by numeric values */
    for (s = strstr(s, ".") + 1; *s != EOS && isdigit(*s); s++);
    return (*s == EOS);
    }
```

```
  /* Choice, Field, and zoom from session names */
  if (strpartcmp(s, "choice.") == 0 ||
      strpartcmp(s, "field.") == 0 ||
      strpartcmp(s, "zoom.from.") == 0)
   {
   /* May only be followed by alpha numerics and a period */
   for (s = strstr(s, ".") + 1; *s != EOS && (isalnum(*s) || *s == '.'); s++);
   return (*s == EOS);
   }

  return (FALSE);
  }

int process_source(void)
  {
  char  destfullpath[PATHSIZE];
  char  objfullpath[PATHSIZE];
  char  src_copy[MAXLINE];
  char  *t, *p;
  char  t1[MAXLINE], t2[MAXLINE];
  int   cleaned;
  int   in_pobj;
  int   error;
  char  save_section[50];
  int   savelvl     = 0;      /* saved path search level of source */

  /* Split source name into component parts */
  strcpy(src_package, substr(src_name, 2, 3));
  strcpy(src_module, substr(src_name, 4, 6));
  strcpy(src_number, substr(src_name, 7, 10));
  sprintf(current_obj, "o%s%s%s", src_package, src_module, src_number);

  /* Locate the actual source file */
  if ((p = search_path(src_name, FALSE, FALSE)) == NULL)
    fatal_msg("source file %s cannot be found in your pacc", src_name);
  strcpy(srcfullpath, p);
  savelvl = lastlvl; /* Save the path search level for our source so */
                     /* we can track down the parent object later.   */


  /* Open source and output destination files */
  if ((srcfile = fopen(srcfullpath, "r")) == NULL)
    fatal_msg("unable to open %s for input", srcfullpath);
  sprintf(destfullpath, "%s.%s", srcfullpath, locasestr(MYNAME));
  if ((destfile = fopen(destfullpath, "w")) == NULL)
    fatal_msg("unable to open %s for output", destfullpath);

  in_pobj = FALSE;
  strcpy(current_section, "");
  strcpy(current_subsection, "");
  genobj = 0;        /* Initially off; use -1 to disable completely */
  cleaned = FALSE;  /* Whether or not we've cleaned old |#POBJ/|#END entries */
  error = FALSE;
  inhibit = FALSE;
  bp_found = FALSE;
  lineno = 0;
```

```
while (fgets(src_line, MAXLINE, srcfile) != NULL)
  {
  chop(src_line);

  /* Strip out all |#pobj/|#end directives atuomatically; we'll readd */
  /* if necessary anyway.                                              */
  strcpy(src_copy, src_line);
  t = strtok(src_copy, WHITESPACE);
  if (t != NULL)
    {
    strcpy(t1, locasestr(t));
    }
  else
    strcpy(t1, "");
  if (strcmp(t1, "|#pobj") == 0)
    {
    in_pobj = TRUE;
    cleaned = TRUE;
    continue;
    }
  if (in_pobj)
    {
    if (strcmp(t1, "|#end") == 0)
      in_pobj = FALSE;
    continue;
    }

  lineno ++;

  /* Found a section/subsection separator */
  if (strcmp(t1, "default:") != 0 && looks_like_section(src_line))
    {
    chop(t1);  /* Strip trailing ":" as it's no longer needed */

    /* If previous section name was functions then we've got an error as */
    /* we've encountered a new section (or sub-section) name.            */
    if (strcmp(current_section, "functions") == 0)
      {
      warning_msg("potential section/subsection found after "
              "'functions'; ignored");
      out_line(FALSE);
      continue;
      }

    /* If we've already scanned a section/subsection and are generating  */
    /* object inheritance code, do so now before going into the next     */
    /* section.                                                          */
    if (current_section[0] != EOS && genobj > 0)
      {
      if (! out_pcall(current_section, current_subsection, FALSE))
        error = 1;
      }

    /* Always reset the inhibit flag on each section/subsection change   */
    /* Must do it here because we don't want to inhibit auto-generated   */
    /* code for undeclared sections/subsections.                         */
    inhibit = FALSE;
```

```
      strcpy(save_section, current_section);

      if (is_section_name(t1))
        {
        /* On section name change, we must output all the remaining        */
        /* subsection calls for interfacing to the parent object.          */
        if (save_section[0] != EOS && genobj > 0)
          out_rest(save_section);

        strcpy(current_section, t1);
        strcpy(current_subsection, ""); /* No subsection or not known yet  */
        }
      else
        strcpy(current_subsection, t1);

      /* Declaration section must be the first one! */
      if (strcmp(current_section, "declaration") == 0 && save_section[0] !=
EOS)
        {
        warning_msg("declaration section must be first section "
                "(no generation)");
        genobj = -1;
        }

      /* If we're entering the before.program section, then              */
      /* output the section name (and comments) and the dll load code    */
      if (strcmp(current_section, "before.program") == 0 && genobj > 0)
        {
        out_line(FALSE);
        out_dllload();
        continue;
        }

      /* If we're entering the function section then output everything   */
      /* from the exported function list that hasn't been done yet.      */
      if (strcmp(current_section, "functions") == 0 && genobj > 0)
        out_all();
      }

    if (strcmp(t1, "|#inhibit") == 0)
      {
      out_line(FALSE);
      inhibit = TRUE;
      continue;
      }

    if (strcmp(t1, "|#call") == 0 && genobj > 0)
      {
      /* The |#call directive was found, if in a valid section/subsection */
      /* attempt to generate the code to execute the parent object --     */
      /* if the parent doesn't have any function for this then we'll just */
      /* ignore this -- it's a comment then.                              */
      out_line(FALSE);
      if (current_section[0] == EOS ||
          strcmp(current_section, "delcaration") == 0)
        {
```

```
      error_msg("|#CALL must be in a non-declaration section");
      error = 1;
      break;
      }
    if (! out_pcall(current_section, current_subsection, TRUE))
      error = 1;

    continue;
    }

/* Found the |#parent directive */
if (strcmp(t1, "|#parent") == 0 || strpartcmp(t1, "|#parent ") == 0)
    {
    /* Can only be defined once and must be defined in the declaration  */
    /* section.                                                         */
    if (genobj == 1)
      {
      error_msg("|#PARENT directive already declared");
      error = 1;
      break;
      }

    if (strcmp(current_section, "declaration") != 0)
      {
      error_msg("|#PARENT directive must be in declaration section");
      error = 1;
      break;
      }

    /* Process the |#PARENT directive */
    genobj = 1;    /* Flag that we're now doing object generation */
    t = nexttok(WHITESPACE);
    if (t != NULL)
      {
      strcpy(parent_obj, t);

      /* Verify user used correct naming convention "oppmmmnnnn" */
      if (strlen(parent_obj) != 10 ||
          parent_obj[0] != 'o'     ||
          ! islower(parent_obj[1]) ||
          ! islower(parent_obj[2]) ||
          ! islower(parent_obj[3]) ||
          ! islower(parent_obj[4]) ||
          ! islower(parent_obj[5]) ||
          ! isdigit(parent_obj[6]) ||
          ! isdigit(parent_obj[7]) ||
          ! isdigit(parent_obj[8]) ||
          ! isdigit(parent_obj[9]))
        {
        error_msg("|#PARENT follwed by invalid object name");
        error = 1;
        break;
        }

      /* Parent must be in same package */
      if (strncmp(current_obj, parent_obj, 3) != 0)
        {
```

```
        error_msg("|#PARENT refers to object in a different package");
        error = 1;
        break;
        }
      }
    else
      {
      sprintf(parent_obj, "o%s%s%s", src_package, src_module, src_number);
      }

    /* Find the parent object; remember to start above the source's      */
    /* current locatin in the VRC pathlist.                              */
    if ((p = search_path(parent_obj, savelvl, TRUE)) == NULL)
      fatal_msg("parent object %s cannot be found in your pacc", parent_obj);
    strcpy(objfullpath, p);
    printf("%s: Inheriting from: %s\n", MYNAME, objfullpath);

    /* Load all the exported section/subsections from the parent object */
    if (! loadparent(objfullpath))
      {
      error = 1;
      break;
      }

    /* Output the required variables declarations for loading the        */
    /* parent dll and making the function calls.                         */
    out_line(FALSE);
    out_startpobj();
    fprintf(destfile, "\tlong\t_pobj_dll_id\n");
    fprintf(destfile, "\tlong\t_pobj_func_id\n");
    fprintf(destfile, "\tlong\t_pobj_err\n");
    fprintf(destfile, "\tstring\t_pobj_path(255)\n");
    out_functiondefs();
    fprintf(destfile, "\t#define _pobj_get(a)\t"
                      "get_function(_pobj_dll_id,a)\n");
    fprintf(destfile, "\t#define _pobj_exe0(a)\t_pobj_err = "
                      "exec_function(_pobj_dll_id,a)\n");
    fprintf(destfile, "\t#define _pobj_exe1(a)\t{ "
                      "_pobj_func_id = get_function(_pobj_dll_id,a)\n");
    fprintf(destfile, "^\t\t\t\t_pobj_err = exec_function(_pobj_dll_id,\n");
    fprintf(destfile, "^\t\t\t\t\t    _pobj_func_id) }\n");
    continue;
    }

  /* Save all comments and blanks; these should be kept with the */
  /* following line. */
  if (*t1 == EOS || *t1 == '|')
    {
    add_comment(src_line);
    continue;
    }

  /* Print the line from the source file as is */
  out_line(FALSE);
  }

/* Output the final section call code */
```

```
  if (current_section[0] != EOS && genobj > 0)
    {
    out_pcall(current_section, current_subsection, FALSE);
    inhibit = FALSE;
    out_rest(current_section);
    }
  /* Output all the rest of the sections exported from the parent object */
  if (genobj > 0)
    {
    inhibit = FALSE;
    out_all();

    out_getfids();
    out_ident();
    }

  /* Output any remaining comments/blank lines */
  out_line(TRUE);

  fclose(srcfile);
  fclose(destfile);

  /* If an error occurred, unlink the temporary object generated source and */
  /* skip calling Baan's std_gen routine (which will also cause the compile */
  /* step to abort as there's no generated source available for it.          */
  if (error)
    unlink(destfullpath);
  else
    {
    /* Ok! Change the source file to be our newly generated file. */
    if (genobj > 0 || cleaned)
      {
      unlink(srcfullpath);
      rename_file(destfullpath, srcfullpath);
      }
    else
      {
      unlink(destfullpath);  /* No changes, scrap this file */
      }
    }

  return (!error);
  }

/*****************************************************************************
 * main function                                                             *
 *****************************************************************************/

void get_base_env(void)
  {
  char *s;
  struct passwd *pw;

  /* Get environment strings */
  if ((s = getenv("BSE")) == NULL)
    fatal_msg("could not determine the BSE");
  strcpy(bse, s);
```

```
  if ((s = getenv("BSE_TMP")) == NULL)
    {
    strcpy(bsetmp, bse);
    strcat(bsetmp, "/tmp");
    }
  else
    strcpy(bsetmp, s);
  if ((s = getenv("USER")) == NULL)
    {
#ifdef WINNT
#else
    if ((pw = getpwuid(getuid())) == NULL)
      fatal_msg("unable to retrieve user information");
    strcpy(user, pw->pw_name);
#endif
    }
  else
    strcpy(user, s);
  }

void get_paths(void)
  {
  #define SPECIALSIZE 4096
  char *s;
  FILE *fd;
  char rec[SPECIALSIZE];
  char pathlist[SPECIALSIZE];
  char pathcode[10];

  if (pacc[0] == EOS)
    {
    if ((s = getenv("PACKAGE_COMB")) == NULL)
      {
      /* Retrieve package combination from user file */
      sprintf(command, "%s/lib/user/u%s", bse, user);
      if ((fd = fopen(command, "r")) == NULL)
      fatal_msg("cannot open user file for %s", user);
      while (fgets(rec, SPECIALSIZE, fd) != NULL)
      {
      chop(rec);
      if (strpartcmp(rec, "pacc:") == 0)
        {
        strtok(rec, ":");
        strcpy(pacc, nexttok(WHITESPACE));
        break;
        }
      }
      fclose(fd);
      }
    else
      strcpy(pacc, s);
    }
  if (pacc[0] == EOS)
    fatal_msg("could not determine what package combination to use");

  /* Retieve all path lists from the fd6.1.xxxxx file */
  sprintf(command, "%s/lib/fd%s.%s", bse, BAANTOOLSVER, pacc);
```

```
  if ((fd = fopen(command, "r")) == NULL)
    fatal_msg("cannot open pacc fd file for %s", pacc);
  while (fgets(rec, SPECIALSIZE, fd) != NULL)
    {
    chop(rec);
    if ((s = strstr(rec, ":")) != NULL)
      {
      *s = EOS;
      s++;
      add_path(rec, s);
      }
    }
  fclose(fd);
  }

void end_program(int exitcode)
  {
  FILE *baanout;
  FILE *engenout;
  FILE *tempfile;
  char rec[MAXLINE];
  char baan4c_work[PATHSIZE];

  if (baan4c_output[0] != EOS)
    {
    /* Reset stdout and stderr to the original output */
    fclose(stdout);
    dup(save_stdout);
    fclose(stderr);
    dup(save_stderr);
    close(save_stderr);
    close(save_stdout);
    tempfile = fdopen(STDOUT, "w");
    memcpy(stdout, tempfile, sizeof(FILE));
    tempfile = fdopen(STDERR, "w");
    memcpy(stderr, tempfile, sizeof(FILE));

    sprintf(baan4c_work, "%s.%d", baan4c_output, getpid());
    sprintf(command, "mv %s %s", baan4c_output, baan4c_work);
    system(command);
    sprintf(command, "mv %s %s", temp_output, baan4c_output);
    system(command);

    /* Transfer the contents of the std_gen output into the work file
       that was created by our program so Baan will pick it all up as
       one output and show this to the developer */

    baanout = fopen(baan4c_work, "r");
    engenout = fopen(baan4c_output, "a");
    while (fgets(rec, MAXLINE, baanout) != NULL)
      {
      fputs(rec, engenout);
      }
    fclose(baanout);
    fclose(engenout);
    unlink(baan4c_work);
```

```
  dispose_sections();
  dispose_paths();
  dispose_comments();

  exit (exitcode);
  }

/* Set stdout to unbuffered; do this by closing it and reopening to the
   same place */
void setnobuf(FILE *buffile, char *mode)
  {
  int hold_fd;
  int orig_fd;
  FILE *newfile;

  orig_fd = fileno(buffile);
  hold_fd = dup(orig_fd);
  fclose(buffile);
  dup2(hold_fd, orig_fd);
  close(hold_fd);
  newfile = fdopen(orig_fd, mode);
  setbuf(newfile, NULL);

  memcpy(buffile, newfile, sizeof(FILE));
  }

int main(int argc, char **argv)
  {
  int i;

  setnobuf(stdout, "w");

  info_msg("Version %s", VERSION);
  info_msg(COPYRIGHT);

#ifdef WINNT
  strcpy(bic_info, "bic_info.exe");
  strcpy(std_gen, "std_genr.exe");
#else
  sprintf(bic_info, "bic_info%s", BAANTOOLSVER);
  sprintf(std_gen, "std_gen%s.real", BAANTOOLSVER);
#endif

  get_base_env();

  /* Scan the command line; we're using std_gen6.1 command line so look */
  /* for "-s source".  We're interested in the source file name and the */
  /* output direction parameters (-qe) that is now used under Baan IVc. */
  strcpy(src_name, "");
  strcpy(baan4c_output, "");
  strcpy(pacc, "");
  for (i = 1; i < argc; i++)
    {
    if (strcmp(argv[i], "-s") == 0)
      {
      i++;
```

```
        if (i < argc)
          strcpy(src_name, argv[i]);
        }
      else if (strcmp(argv[i], "-pacc") == 0)
        {
        /* Use this for the package combination name */
        i++;
        if (i < argc)
        {
        strcpy(pacc, argv[i]);
        }
        }
      else if (strcmp(argv[i], "-qe") == 0)
        {
        i++;
        if (i < argc)
          {
          int dupout;

          strcpy(baan4c_output, argv[i]);
          sprintf(temp_output, "%s/tmp%s.%d", bsetmp, MYNAME, getpid());

          /* Redirect stdout and stderr to a file so it can be placed in the */
          /* output file for Baan to display to the developer properly.      */
          save_stdout = dup(STDOUT);
          save_stderr = dup(STDERR);
          if (freopen(temp_output, "w", stdout) == NULL)
            fatal_msg("unable to reopen stdout");
          dup2(STDOUT, STDERR);

        info_msg("Version %s", VERSION);
        info_msg(COPYRIGHT);
          }
        }
      }
    if (src_name[0] == EOS)
      fatal_msg("source file name not found on command line");

    get_paths();
    if (process_source())
      {
      int status;
      pid_t child;
      char orig_name[FILENAMESIZE];

      /* Call Baan's std_gen program to process the source file */
      if (genobj > 0)
        info_msg("Executing std_gen on %s source", MYNAME);
      else
        info_msg("Executing std_gen on normal source");

      if ((child = fork()) == 0)
        {
        sprintf(command, "%s/bin/%s", bse, std_gen);
#ifdef WINNT
        strcpy(orig_name, "stdgen.exe");
#else
```

```
        sprintf(orig_name, "std_gen%s", BAANTOOLSVER);
#endif
        argv[0] = orig_name;
        execvp(command, argv);
        fatal_msg("unable to execute %s", std_gen);
        exit(1);
        }
      waitpid(child, &status, 0);
      end_program(status/256);
      }

  end_program(1);
  }
```

Unlock the full potential of your

Baan ERP system with QKEY...

QUALITY CONSULTANTS, INC. B*aa*N

An Enterprise Resource Planning solution is no small investment. So it had better open some new doors. Above all, your ERP system should enable you to extend your capabilities, expand your horizons and respond to a dynamic environment. It must be robust and flexible, and leave you in control.

Quality Consultants Inc. (QCI) is committed to providing innovative tools and proven expertise that allow businesses to optimize the performance of their ERP systems. Drawing upon our in-depth experience in ERP implementation and systems integration, QCI has developed a revolutionary development tool – QKEY – that lets you modify and enhance your Baan ERP system *without using Baan source code.*

QKEY Object Applet Integrator for Baan adds a new dimension of responsiveness to ERP and lets you, not your software, set the agenda and the pace. Acting as a bridge between a manufacturer's specific needs and the capabilities of the Baan system, QKEY is a unique software utility that makes customization and upgrades easier, faster and more economical. You can enhance functionality and refine your system to meet the specific demands of your business. Reduce testing time during the upgrading process. Even better, all your developers need is Baan Tools knowledge, freeing you from the need to hire expensive source code specialists.

At the core of QKEY is a preprocessor for Baan source scripts. Developers can use QKEY to manipulate 4GL scripts to enhance current Baan sessions without having the source code available. QKEY makes this possible by allowing the user to view each Baan standard session as an object, and by permitting the overriding or extension of this object's associated events (or *public methods* in Object Oriented Programming terminology). The result? You can create customized 4GL objects that contain only your new code so that when a Baan patch or Version Release Control (VRC) is installed your changes can be reused automatically. In most cases, there is no need to recompile customizations. Even in installations that have already purchased Baan's source code, these features can save you considerable time and money.

Significantly lowered programming costs, ease of implementation and reduced development time are just the beginning. With compressed integration cycles and an information technology infrastructure that's customized to your unique needs, you're free to meet the distinctive demands of your supply chain faster and more accurately. In a marketplace where speed and agility are nothing short of critical, QKEY's benefits make perfect sense. What's more, QKEY is priced low to guarantee you a rapid return on investment.

- No need for Baan source code when making enhancements.
- Easy to install and easy to use.
- Integrates seamlessly with the Baan development environment and does not interfere with standard development techniques.
- Only new changes are stored in customized scripts.
- QKEY generator automatically searches the VRC paths to find the standard object.
- Intermediate QKEY objects in the VRC tree are skipped.
- Dynamically searches for and loads the standard object when a session starts.
- Patched objects are automatically used if found first in the VRC path.
- Works with 4GL scripts.
- Baan debugger can still be used.

"A major cu. omization for us was pricing. Our costs and selling prices are driven by commodity prices, which change daily. QKEY allowed us to design, develop and implement our unique pricing methodology without modifying Baan source code. This was very important to us because we wanted to use a system that allowed us to benefit from upgrades and still customize the system to fit our business.

"QKEY has enhanced our ability to migrate from the legacy system to Baan, and without it, we could not have delivered even a partial implementation in our time frame. I am very satisfied with the product."

Joy Conner
Programmer Analyst
JC Nordt

Quality Consultants Inc. (QCI) specializes in the implementation and support of Baan's Enterprise Resource Planning software solutions. QCI is a licensed Baan Service Partner and a member of the Process Technology Group of companies.

"QKEY is an excellent product that makes it easy to modify scripts without purchasing Baan source code. This saves our company money and allows us to make the modifications necessary to fit our business. With QKEY, we are able to make enhancements to Baan scripts, including fields and specific sections within the script. We are very satisfied with QKEY."

Phil Chesher
Systems Analyst
The Andersons

CLAIMS

WHAT IS CLAIMED IS:

1. An apparatus for preprocessing existing source scripts, comprising:

a) a computer having a memory storage device and a microprocessor;

b) an operating system stored in said memory storage device;

c) means for viewing an object; and,

d) means for permitting the overriding of the associated events of said object.

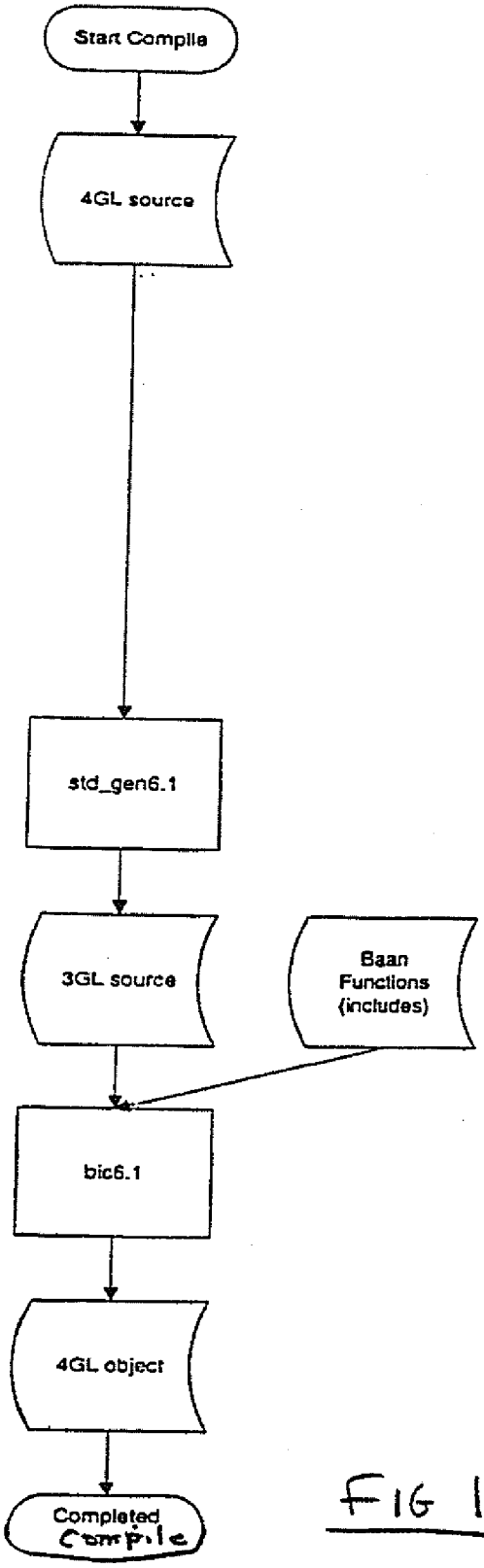


Fig 1a

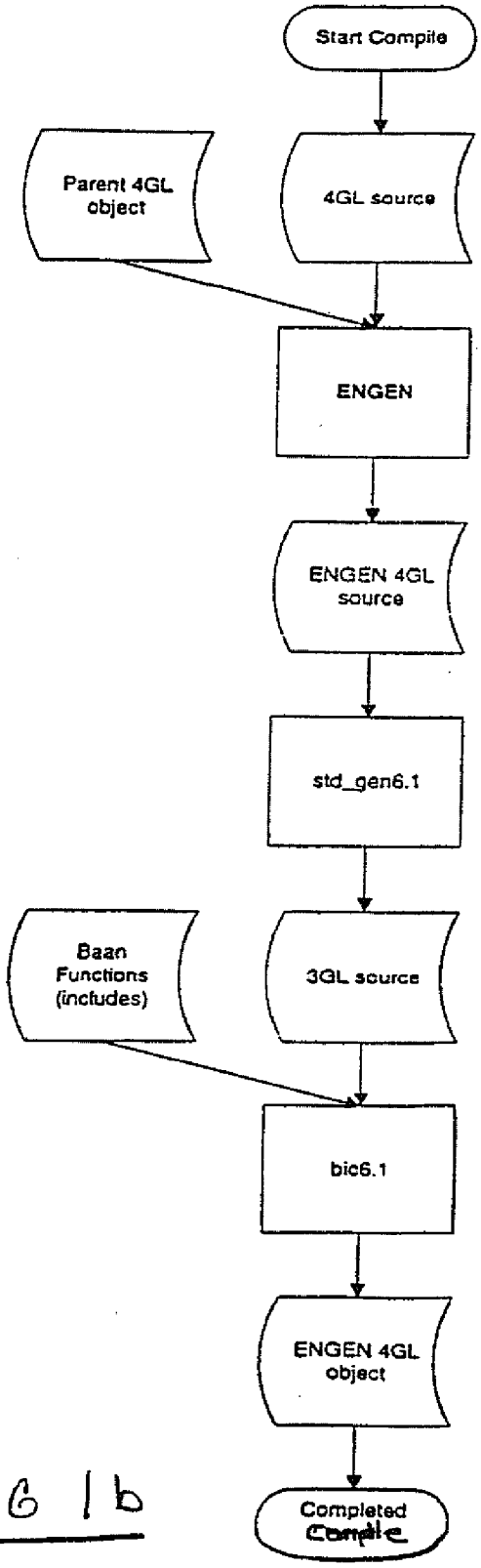


Fig 1b

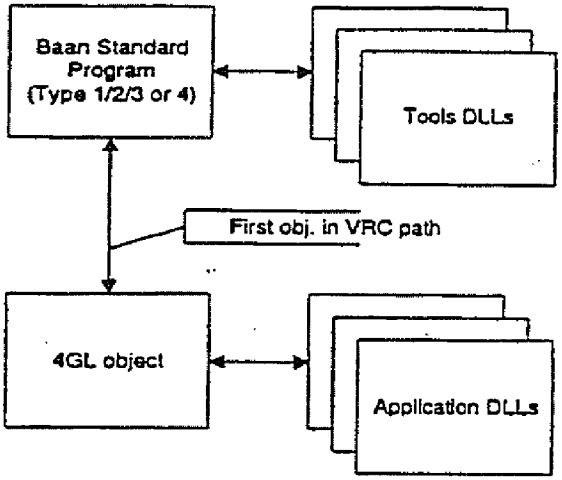


FIG. 2a

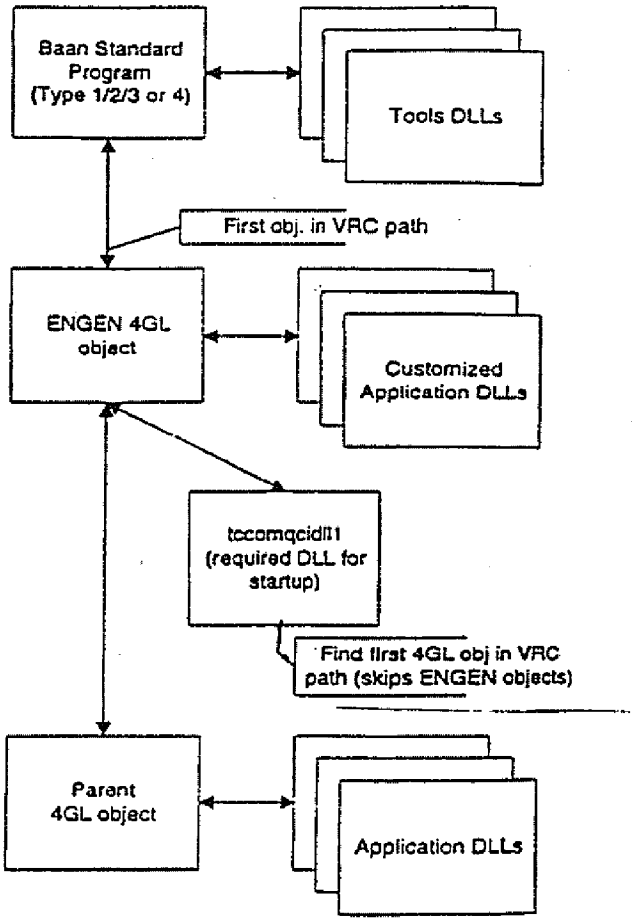


FIG. 2b

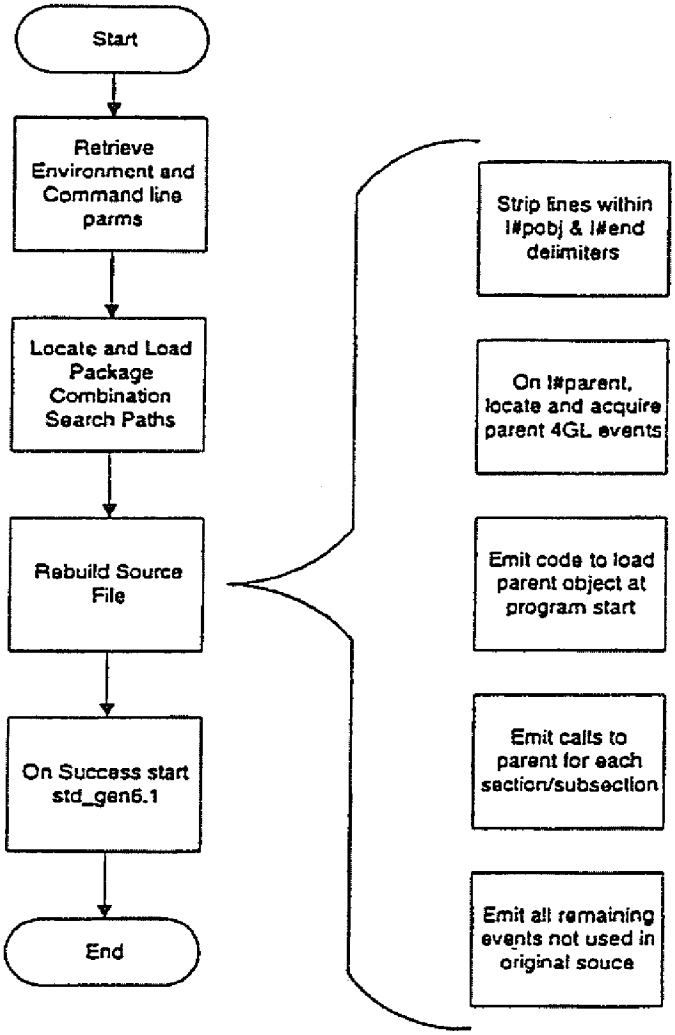


FIG 3

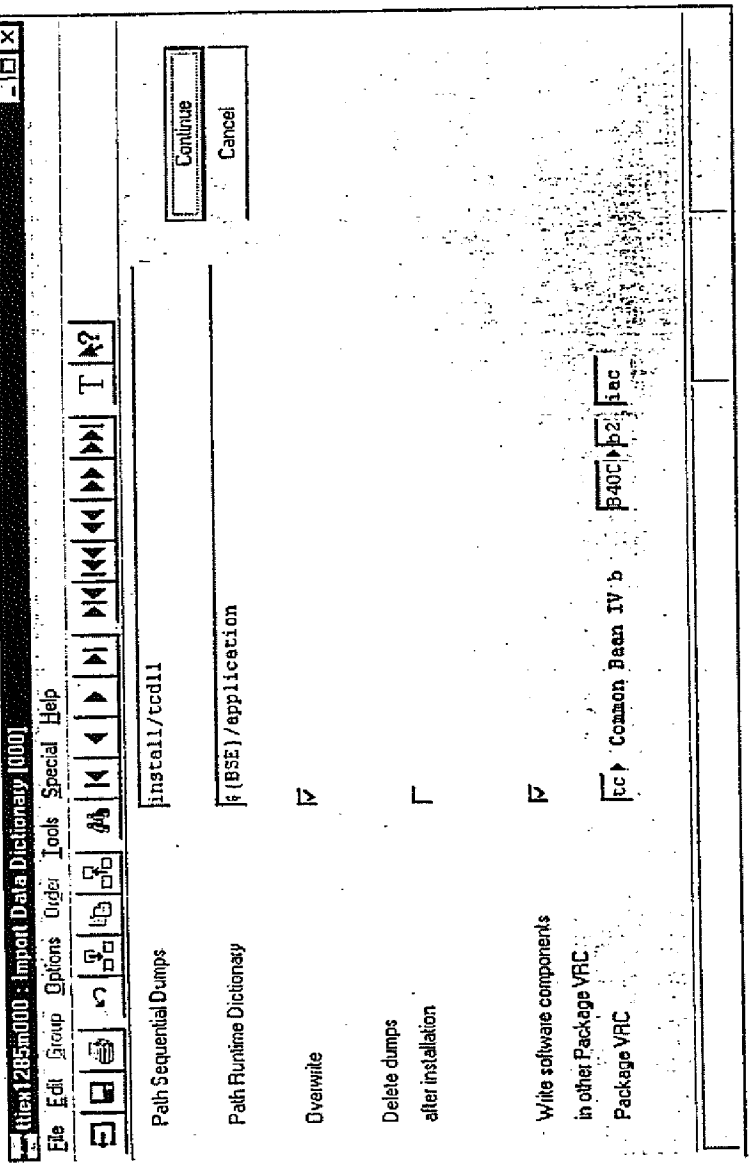


FIG. 4

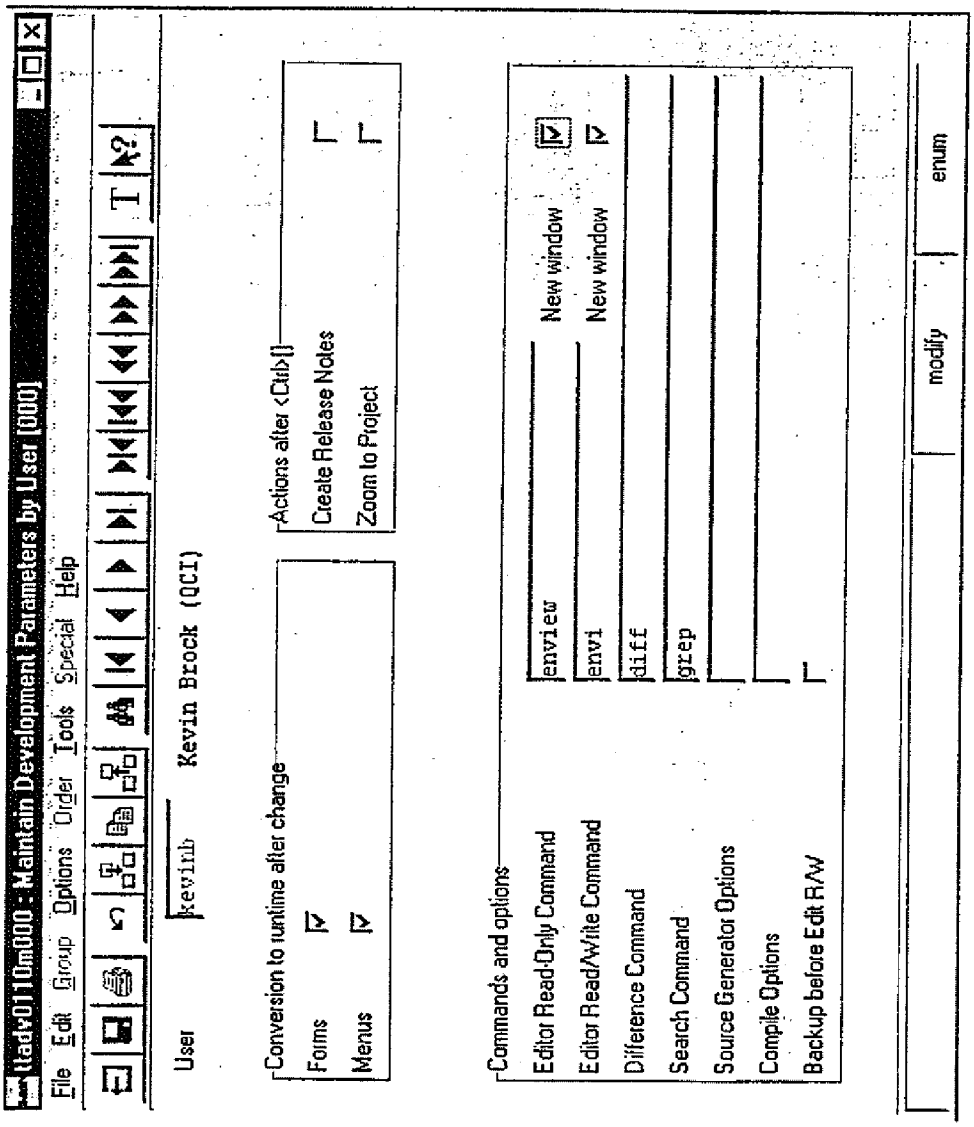


FIG. 5

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 99/12075

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	QUALITY CONSULTANTS: "QKEY 2.10 User and Technical Guide" QKEY ENCAPSULATED GENERATOR FOR BAAN, 12 April 1998 (1998-04-12), page 1 XP002124554 the whole document	1
X	US 5 754 862 A (JONES DAVID T ET AL) 19 May 1998 (1998-05-19) column 2, line 62 -column 3, line 10 column 7, line 65 -column 9, line 10 column 10, line 23 -column 11, line 35	1

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 December 1999	22/12/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Bijn, K

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

ernational Application No
PCT/US 99/12075

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5754862 A	19-05-1998	US 5410705 A	25-04-1995
		US 5297284 A	22-03-1994
		US 5854931 A	29-12-1995